НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК 004.75

«До захисту допущено»

Завідувач кафедри СПСКС

_____ В.П.Тарасенко
(підпис) (ініціали, прізвище)

"___"_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія
Комп'ютерні системи та компоненти

на тему: Система розподілу навантаження на сервіси хмарної інфраструктури

Виконав: студент II курсу, групи КВ-71мп
(шифр групи)

Ралко Денис Андрійович
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник к.т.н., доцент Сапсай Т.Г.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

Національний технічний університет України
“Київський політехнічний інститут
імені Ігоря Сікорського”

Факультет прикладної математики

Кафедра системного програмування і
спеціалізованих комп‘ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність 123 “Комп’ютерна інженерія”
Комп‘ютерні системи та компоненти

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____В.П.Тарасенко
“___”_____2018 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ
Ралко Денису Андрійовичу

1. Тема дисертації: Система розподілу навантаження на сервіси хмарної інфраструктури,
науковий керівник дисертації: доцент кафедри СПіСКС, к.т.н., доцент Сапсай Т.Г.,
затверджені наказом по університету від “30” жовтня 2018 року № 4030-с.
2. Термін подання студентом дисертації: “7” грудня 2018 р.

3. Об’єкт дослідження: хмарні інфраструктури.

4. Предмет дослідження: алгоритм розподілу навантаження у хмарній інфраструктурі.

5. Перелік задач, які потрібно вирішити:
- Дослідження та аналіз існуючих методів та алгоритмів для балансування навантаження у хмарній інфраструктурі;
- Розробка моделі системи балансування навантаження;
- Розробка алгоритмів створення нових серверів та визначення навантаженості системи;

- Програмна реалізація системи;
- Тестування системи.

6. Перелік ілюстративного матеріалу:

- Презентація.

7. Перелік публікацій:
- XI конференція молодих вчених «Прикладна математика та комп'ютинг» ПМК-2018-2;
- IV Міжнародна науково-технічна Internet-конференція «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами».

8. Дата видачі завдання: "4" вересня 2017 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вибір теми магістерської дисертації	03.09.2017	
2.	Вивчення та аналіз існуючих матеріалів	03.11.2017	
3.	Проходження практики	03.09.2018	
4.	Написання першого розділу	03.07.2018	
5.	Написання другого розділу	03.08.2018	
6.	Написання третього розділу	03.09.2018	
7.	Оформлення текстової частини магістерської дисертації	03.10.2018	
8.	Попередній розгляд магістерської дисертації на засіданні кафедри	26.11.2018	

Студент _____ Ралко Д. А.
(підпис)

Науковий керівник дисертації _____ Сапсай Т. Г.
(підпис)

РЕФЕРАТ

Актуальність теми. Хмарні обчислення – це модель надання зручного мережевого доступу до обчислювальних ресурсів, що конфігуруються, та можуть бути швидко надані і звільнені з мінімальними зусиллями по управлінню. Актуальним є аналіз можливостей використання хмарних технологій в усіх сферах та розробка основних принципів підвищення ефективності обробки інформації та забезпечення їх надійного зберігання у хмарі. Одним з методів підвищення ефективності є розробка балансування навантаження. На даний час існує багато систем розподілення навантаження, що працюють на базі різних алгоритмів. Але наразі немає такого, що б міг збільшувати пропускну здатність для структур, що використовують системи обміну повідомленнями. Тому наразі, дана тема магістерської дисертації є актуальна.

Об'єктом дослідження є система розподілення навантаження на сервіси хмарної інфраструктури.

Предметом дослідження є методи та алгоритми для розподілення навантаження на модулі у хмарному середовищі.

Методи досліджень – порівняльна характеристика найбільш вживаних методів розподілення навантаження на сервіси хмарної інфраструктури, порівняння алгоритмів за швидкодією і пропускною спроможністю на однорідних типах задач. Визначення недоліків та переваг різних способів балансування навантаження, та пропонування свого рішення покращення часу обробки даних.

Мета і задачі дослідження: розробка системи розподілу навантаження на сервісах хмарної інфраструктури Google Cloud Platform, що обмінюються даними між собою за допомогою системи обміну

повідомленнями RabbitMQ. Для цього визначено завдання, які вирішуються в роботі:

1. Проведення аналізу існуючих методів і алгоритмів, які забезпечують вирішення задач балансування навантаження в хмарних інфраструктурах.
2. Розглянути і порівняти способи балансування навантаження.
3. Дослідити системи вирівнювання навантаження
4. Розробити систему розподілення навантаження на сервіси хмарної інфраструктури.
5. Експериментальне дослідження характеристик створеної системи розподілення навантаження.

Наукова новизна одержаних результатів полягає в наступному:

1. Проведено порівняння існуючих методів балансування навантаження та проаналізовано правила та способи балансування навантаження.

2. Запропоновано модифікований спосіб динамічного балансування навантаження з масштабуванням на хмарній інфраструктурі, який відрізняється від відомих способів можливістю обробляти задачі, що знаходяться в чергах брокеру RabbitMQ.

3. Розроблена система розподілу навантаження на сервіси хмарної інфраструктури на основі модифікованого способу динамічного балансування навантаження.

Практична цінність одержаних в роботі результатів дозволяють зменшити час обробки завдань, що надходять з системи обміну повідомлень на модуль.. Також проведені експериментальні дослідження, які показали, що час обробки завдань при використанні модифікованого способу балансування навантаження зменшується.

Апробація роботи. Основні положення і результати роботи представлені та обговорені на:

- IV Міжнародна науково-технічна Internet-конференція «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами».
- XI конференція молодих вчених « Прикладна математика та коип‘ютинг ПМК-2018-2.

Публікації. За темою досліджень опубліковано 2 наукові праці- тези доповідей на конференціях.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, трьох розділів, висновків та додатків.

У вступі зроблено оцінку сучасного стану хмарних інфраструктур, подано загальну характеристику роботи, обґрунтовано актуальність напряму дослідження, сформульовано мету та задачі досліджень.

У першому розділі наведено огляд сучасних технології хмарних обчислень, опис ознак хмарних обчислень, проведено аналіз переваг та недоліків хмарної інфраструктури, розглянуті можливості гнучкого масштабування.

У другому розділі розглянуто способи балансування та системи вирівнювання навантаження, проаналізовано їх характеристики, особливості структури та функціональні можливості.

У третьому розділі розроблено систему розподілу навантаження на сервіси хмарної інфраструктури на основі модифікованого способу динамічного балансування.

У висновках представлені результати проведеної роботи.

Робота представлена на 81 аркушах, містить 24 рисунків, 3 таблиці та посилання на список використаних літературних джерел з 26 найменувань.

Ключові слова: хмарні інфраструктури, балансування навантаження, сервери, система обміну повідомлень, Google Cloud Platform.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API – Application programming interface, набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

CORBA – The Common Object Request Broker Architecture, технологічний стандарт розробки розподілених застосунків.

DDoS – Denial-of-service attack, напад на комп'ютерну систему з наміром зробити комп'ютерні ресурси недоступними користувачам, для яких комп'ютерна система була призначена.

DNS – Domain Name System, ієрархічна розподілена система перетворення імені хоста в IP-адресу.

HAProxy – Hidh Availability Proxy, вільне програмне забезпечення, проксі-сервер і балансувальник наватаження в системах з високою доступністю.

HTTP – Hypertext Transfer Protocol, протокол передачі даних, що використовується в комп'ютерних мережах.

IDC – International Data Corporation, є постачальником маркетингової інформації, консультаційних послуг і заходів для ринків інформаційних технологій, телекомунікацій і споживчих технологій.

IPv6 – Internet Protocol version 6, нова версія IP-протоколу — IP версії 6

IT – інформаційні технології.

JSON – JavaScript Object Notation, це текстовий формат обміну даними між комп'ютерами.

QOS – Quality of service, якість послуг, які надає комунікаційна мережа.

RMI – Remote Method Invocation, програмний інтерфейс виклику віддалених методів у мові Java.

ROI – Return on investment, фінансовий коефіцієнт, який ілюструє рівень прибутковості та збитковості бізнесу.

RPC – Remote Procedure Call, протокол, що дозволяє програмі, запущеній на одному комп'ютері, звертатись до функцій (процедур) програми, що

виконується на іншому комп'ютері, подібно до того, як програма звертається до власних локальних функцій.

SCP – Secure Copy Protocol, протокол для безпечного копіювання файлів між локальним та віддаленим хостом або між двома віддаленими хостами.

SDK – Software development kit, набір із засобів розробки, утиліт і документації, який дозволяє програмістам створювати прикладні програми за визначеною технологією або для певної платформи.

SLA – Service-level agreement, угода між постачальником послуг і користувачем про рівень послуг.

SOAP – Simple Object Access Protocol, протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, базується на форматі XML.

SSL – Transport Layer Security, криптографічний протокол, що надає можливості безпечної передачі даних в Інтернет для навігації, отримання пошти, спілкування, обміну файлами.

TCP/IP – набір протоколів мережі Інтернет. Назва походить від назви стрижневих протоколів мережі Інтернет — IP (англ. Internet Protocol — «міжмережевий протокол») і TCP (англ. Transmission Control Protocol — «протокол керування передаванням»). Фактично це систематизований стек протоколів, що поділяється на чотири рівні, які корелюються з еталонною моделлю OSI.

WSDL – Web Services Description Language, мова визначення інтерфейсу веб-сервісу заснована на XML, що описує функціональність веб-сервісу і спосіб доступу до нього.

XML – Extensible Markup Language, запропонований консорціумом World Wide Web стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет.

ОС – операційна система.

ВСТУП

Хмарні інфраструктури стали новим напрямом розвитку великомасштабних розподілених систем. Вони визначаються як «модель для забезпечення зручного мережевого доступу на вимогу до загальних ресурсів (наприклад, мереж, серверів, сховищ, додатків і сервісів), які можуть бути швидко підготовлені і надані з мінімальними експлуатаційними витратами або взаємодією з постачальником послуг». Хмарні обчислення - це концепція паралельних і розподілених обчислень, які в міру необхідності використовують ресурси, такі як апаратне забезпечення, програмне забезпечення та інформацію на комп'ютерах або інших пристроях.

Згідно з дослідженнями IDC (International Data Corporation), створення інфраструктури для підтримки хмарних обчислень, на сьогодні складає більше третини всіх витрат на IT (Інформаційні технології) у всьому світі [11]. Однак витрати на традиційну IT-інфраструктуру продовжують знижуватися, оскільки обчислювальні навантаження продовжують переміщатися в хмару, будь-то загальнодоступні хмарні сервіси, пропоновані постачальниками або приватні хмари, створені самими підприємствами.

Балансування навантаження є життєво важливою частиною для паралельних і розподілених систем. Використання систем балансування навантаження забезпечує підвищення загальної продуктивність, обчислювальних ресурсів, поліпшуючи управління споживанням енергії, покращує якість обслуговування при розподілі хмарних сервісів, підтримуючи стабільність системи за рахунок розподілу та управління робочими навантаженнями в хмарних інфраструктурах.

Однією з найважливіших задач при використанні хмарного середовища є розподіл навантаження на сервери. Це відноситься до розподілу клієнтських запитів на декількох серверах додатків, що

працюють в середовищі. Динамічне балансування обчислювального навантаження в хмарній інфраструктурі дозволяє не тільки максимально підвищити продуктивність обробки запитів, а і забезпечити надійність розподілених додатків. Основними перевагами балансування навантаження в хмарних інфраструктурах є можливість масштабування, зниження вартості обробки інформації за рахунок використання розподілених ресурсів та забезпечення необхідного сервісу для конкретних запитів, що суттєво впливає на ефективність надання послуг. Метою даної роботи є розробка системи розподілу навантаження на сервіси хмарної інфраструктури на основі модифікованого способу динамічного балансування навантаження з масштабуванням на хмарній інфраструктурі, який відрізняється від відомих способів можливістю обробки задач, що знаходяться в чергах брокера RabbitMQ; проведення експериментальних досліджень для оцінки ефективності роботи системи.

1. АНАЛІТИЧНИЙ ОГЛЯД ХМАРНОЇ ІНФРАСТРУКТУРИ

1.1 Сучасні технології хмарних обчислень

Суттєвим аспектом хмарних обчислень стало застосування можливостей керувати ресурсами, тобто всім, що бере участь при обробці даних, які забезпечують з'єднання в єдину систему ресурсів: засобів збереження даних, серверів, устаткувань для мережі, сервісів, файлообмінників тощо. Це все, що бере участь в локально та глобально розподілених середовищах.

Електронна обчислювальна машина, що бере участь в обробці даних під час виконання програми розподілення, з'єднується з Інтернет – сервісом за протоколом HTTP. Без додаткових налаштувань обмін даними в мережі за іншими портами ускладнюється, оскільки більшість проксі-серверів та брандмауерів таке з'єднання фільтрують.

Ще до того, як з'явилася веб-служба, світ вже був знайомий з іншими технологіями. Це давало можливість передавати сигнал на відстані, незалежно від того, знаходиться комп'ютер в іншому приміщенні, чи навіть країні, оскільки одна програма могла викликати будь-який інший ресурс. Виклик віддалених процедур отримав назву Remote Procedure Calling (RPC), серед яких можна виділити Common Object Request Broker Architecture (CORBA) та Remote Method Invoking (RMI). Основною метою веб-служби було створити такий Remote Method Invoking, який би зміг співдіяти з пакетом HTTP [11].

Спочатку необхідно описати формат виклику в XML такий, як Simple Object Access Protocol (SOAP), протокол обміну структурованими повідомленнями в розподіленому обчислювальному середовищі. Дані відправляються по HTTP, де на початку відбувається перетворення в XML опис Simple Object Access Protocol. Далі в HTTP пакеті надсилається на інший персональний комп'ютер по TCP/IP. WSDL (англ. Web Services Description Language) – являє собою мову опису веб-сервісів і доступу до них, заснований на форматі XML. WSDL не тільки описує мережевий

інтерфейс веб-служби, а й скориставшись методами сервісу, має можливість дізнатися про способи доступу до інформації про послуги.

Тепер можна розглянути питання використання веб-сервісу, який має сервер і клієнта. Кінцевою точкою, куди від клієнта доходять SOAP-повідомлення, є сервер.

Головні етапи здійснення:

- описання інтерфейсу веб-служби;
- втілення інтерфейсу;
- запуск веб-служби;
- створення клієнта;
- виклик потрібного методу веб-служби.

За допомогою стандартних протоколів мережі Інтернет, використовуються функції додатків, які складають єдину концепцію веб-сервісу. Їх втілення здійснюється за участю стандартизованих технологій – World Wide Web Consortium (W3C).

Шляхом перерозподілу обчислювальних ресурсів між задачами, можна досягти підвищеної продуктивності обчислень до задовільного показника. Наразі, це є актуальним питанням, оскільки у користувачів все частіше виникає потреба зневажливого ставлення до ресурсів, які робоча станція не може забезпечити.

1.2 Опис ознак хмарних обчислень

Завдяки швидкому росту хмарного середовища, стає можливим швидкісне зростання його ефективності, економічність коштів та розвиток інновацій уряду, економіки і т.д. Основними ознаками хмарних обчислень є можливість масштабувати, зберігати дані та здатність швидко виконувати певні операції. Користувач при цьому може не керувати більшістю технологій.

Для реалізації потенціалу хмарних обчислень потрібно:

- покращити впевненість кожного користувача;

- опрацьовувати стереотипи та механізми роботи;
- удосконалити на законодавчому рівні зростання вкладів у хмарне середовище.

Користувачі повинні бути впевнені у тому, що запускаючи програми та зберігаючи у хмарі свої персональні дані, вони будуть захищені та не розголошені. Так само, це актуально і для приватних користувачів.

Для того, щоб досягти безпеки хмарного середовища на гідному рівні, постачальники таких сервісів повинні:

- періодично тестувати рівень безпеки;
- здійснювати контроль за доступом шляхом ідентифікації, автентифікації;
- використовувати тільки перевірені та широковідомі методи контролю.

1.3 Особливості роботи

Монітор віртуальної машини (гіпервізор) – це програма, або обладнання процесора, яка надає можливість паралельно виконувати декілька операційних систем на одному хост-комп’ютері (host computer). Віртуальними машинами при цьому використовується ресурс хост-комп’ютера, такий як пам'ять, сховище та інше обладнання. Використання основних показників при цьому зростає.

Монітор віртуальної машини відокремлює всі операційні системи (ОС) від хост-комп’ютера і служить для задоволення вимог гостьових ОС. Не дивлячись на те, що віртуальні машини паралельно працюють на одному основному комп’ютері, вони між собою не зв’язані. Тому у разі виходу із ладу однієї з віртуальних машин, на роботу інших це впливати не буде, вони продовжать роботу у звичному режимі.

Монітори віртуальної машини розділяються на два типи. До першого (автономного) типу можна віднести системи Microsoft Hyper-V, VMware ESXi, Citrix XenServer. Для нього характерна робота в автономному

режимі безпосередньо на обладнанні, а тому від базової оперативної системи він не залежить. До другого типу належать системи Parallels Desktop і VMware Player. Він взаємодіє з центральною частиною основної операційної системи, проте працює поверх апаратного забезпечення, на окремому рівні. А операційна працює як другий рівень.

1.4 Форми розміщення хмарної інфраструктури

Хмарні послуги (ХП) – послуги з наданням хмарних ресурсів за допомогою технологій хмарних обчислень. В системі «4-3-2», цифра 3 визначає моделі ХП, які можуть працювати як відокремлено, так і разом. Хмарні сервіси забезпечують швидке масштабування, оскільки мають можливість швидко розгортати і розповсюджувати. Послуга має необмежену кількість та характер, що робить її доступною для користувача [12].

Виділяють такі форми хмарних послуг:

- Software as a Service (SAAS) – вид послуг, коли користувачам надається готове програмне забезпечення через підписку на програмне обслуговування, яке забезпечується провайдером.
- Platform as a Service (PAAS) – користувачеві надано доступ до використання інформаційно-технологічних платформ, таких як операційна система, управління базами даних, розміщених у хмарному середовищі.
- Infrastructure as a Service (IAAS) - користувачам на підставі здійсненої оплати надано інформаційно-технологічні ресурси, а саме віртуальні сервери, в яких задано обчислювальна потужність, операційна система і доступ до мережі.

Далі представлено особливості кожної із хмарних послуг.

Infrastructure as a Service - модель обслуговування хмарних обчислень (ХО), при якій користувачі використовують власні платформи в інфраструктурі постачальника (Рис.1.1).

Основними привілегіями (IAAS) є:

- можливість оплати користувачами за запитами;
- масштабування інфраструктури в залежності від пам’яті та необхідної потужності комп’ютера;
- відсутність як таких витрат на придбання та обслуговування персонального ПЗ;
- відсутність єдиної точки відмови, оскільки всі дані перебувають у хмарі;
- адміністрування задач підтримує віртуалізацію серверу. При цьому можливе одночасне виконання іншої роботи.

Рисунок 1.1 – Цикл задач моделі Infrastructure as a Service

Platform as a Service – модель обслуговування хмарних обчислень, при якій користувач створює розробку і керує додатками, в межах хмарного середовища (Рис 1.2). Основними привілегіями (PAAS) являються:

- розробляти та розміщувати додатки, тестувати можна у тому ж середовищі;

- постачальник здійснює контроль за захистом, ОС, серверними програмами та резервною копією;
- навіть у разі віддаленої роботи співробітників, спільна робота надає дієві результати.

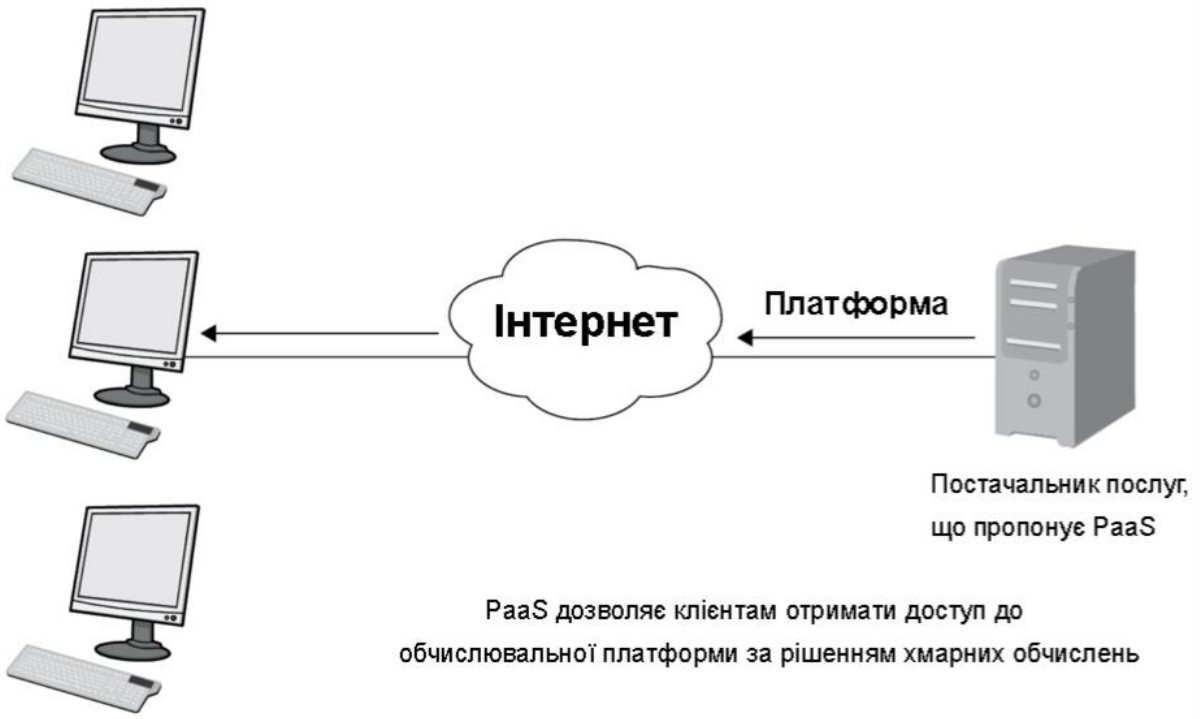

Рисунок 1.2 – Схема роботи моделі Platform as a Service

Software as a Service – модель обслуговування хмарних обчислень, при якому додаток не встановлюється споживачем на персональний пристрій, а доступ до додатків, які перебувають у хмарній мережі, здійснюється через application programming interface (API) або веб-інтерфейс (Рис 1.3). Дане хмарне рішення дає можливість користувачам, незалежно від локального місцезнаходження спільно співпрацювати, здійснювати зберігання та аналіз інформації.

Основними привілеями (SAAS) є:

- доступ до програмного забезпечення через підписку на програмне обслуговування;
- керування та встановлення здійснюється постачальником;

- інформація зберігається віддалено, що унеможливлює втрату даних;
- доступ до інформації з будь-якого пристрою, підключеного до мережі Інтернет.

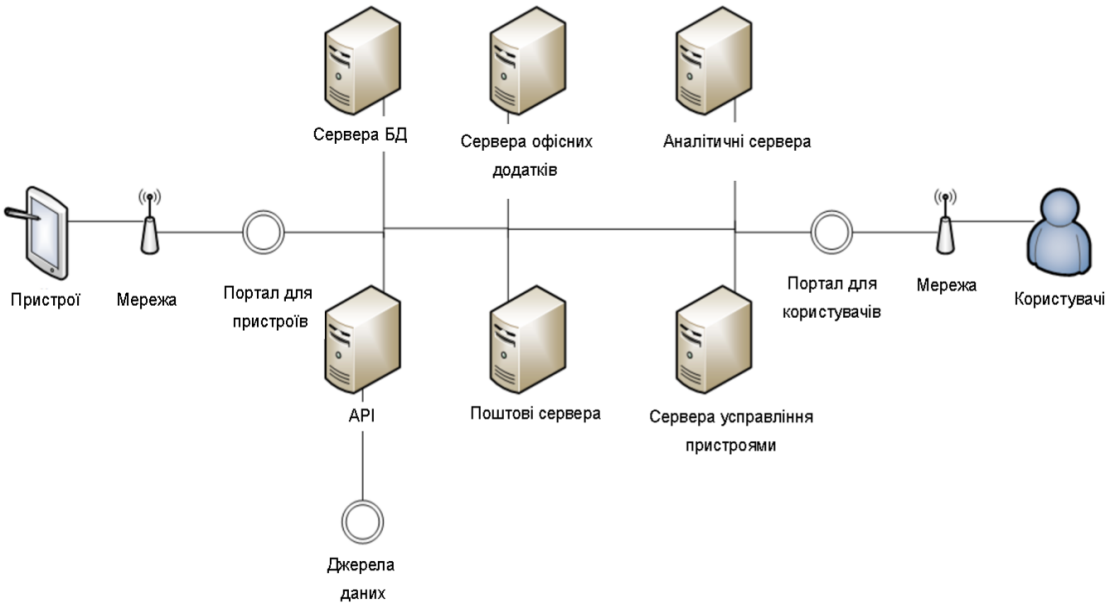

Рисунок 1.3 – Схема роботи моделі Software as a Service

Для споживачів усі три методи можуть бути дієвими та корисними при використанні як у власних цілях, так і для роботи. Кожен із методів має свої як переваги, так і недоліки, розділяючи відповідальність постачальника послуг та користувача. Відповідальність провайдера відображено в табл 1.1.

Використання хмарного сервісу повинно бути захищене, оскільки передаючи на обробку свої дані інтернет-провайдеру, користувач повинен бути впевненим, що його дані надійно захищені та робота з сервісом безпечна.

На аналізі ризиків ґрунтується не тільки вибір методів захисту, але і принципи вибору бізнес-процесів, які загружаються у хмарне сховище. Таким чином, користувач являється первинною ланкою у захисті даних у хмарі, а провайдер кінцевим.

Таблиця 1.1. – Розмежування хмарних послуг

	SAAS	PAAS	IAAS
Мережі	+	+	+
Програми	+	_	_
Сховище	+	+	+
Дані	+	_	_
Сервер	+	+	+
Віртуалізація	+	+	+
ОС	+	+	_
ПЗ	+	+	_
Середовище виконання	+	+	_

1.5 Переваги та недоліки хмарної інфраструктури

Наразі, більшість компаній ще не переносять свої інформаційні технології до хмарних середовищ. Перш за все це пов’язано з ризиками щодо безпеки користування. Однак є багато плюсів, які все ж спонукають до використання хмарного середовища. Серед ризиків, пов’язаних з ХО можна виділити:

1. Віддаленість хмарних серверів, у зв’язку з чим користувач не може контролювати свої персональні дані. При цьому хмарний сервер знаходиться не в центрі обробки даних, як це відбувається зазвичай, а віддалено.
2. Загроза безпеці. Кіберзлочинність не стоїть на місці, постійно з'являються шляхи впливу на хмарне середовище. Це загрозливі програми, різноманітні віруси, хакерські атаки, такі як, наприклад, Distributed Denial of Service. Все це підриває безпеку і створює загрозу для підприємництва.

3. Сприяння. Недобросовісні працівники можуть передати інформацію конкурентам, оскільки аналогічні послуги пропонують не всі постачальники хмарного хостингу.
4. Підключення. При не стабільному підключенні до мережі Інтернет, існує загроза втрати інформації, що може негативно вплинути на бізнес.
5. Поширення інформації. Інформація, яка знаходиться у хмарному середовищі, може бути оприлюднена правоохоронними органами, що пов’язано із сьогоднішніми подіями у світі.

До переваг хмарного обчислення відносяться:

1. Економічна вигідність. Користувачі, де б вони не знаходились, мають можливість отримувати доступ до своїх даних. Після використання однієї із моделей хмарного середовища, вони отримують рахунок, тому можна самостійно контролювати витрачені ресурси.
2. Безпечність. Провайдери хмарного середовища шифрують дані, що ускладнює несанкціоноване вторгнення.
3. Оптимізація роботи. Працівники компанії, можуть обмінюватись даними, навіть перебуваючи у різних місцях, без погіршення роботи.
4. Випробуваний. Працівники бізнесу можуть бути впевненими, що навіть поломка одного із серверів, збереже інформацію, оскільки доставляється з інших веб-ресурсів у кластері.
5. Самостійне оновлення. Оскільки сервери хмарного середовища можуть перебувати у різних місцях, система оновлюється автоматично. Це забезпечує не тільки безпеку, а й відсутність необхідності самостійного оновлення.

1.6 Можливість гнучкого масштабування

Заздалегідь, важно передбачити, яка кількість запитів передаватиметься на сервер. Класична модель балансування та

впровадження за допомогою серверів навантаження значно відрізняється від балансування хмарного середовища.

Надаючи свій винятковий комплекс завдань, забезпечуються нові перспективи. Основною проблемою хмарних обчислень, є все ж таки балансування навантажень, коли одні вузли перевантажені, а інші не працюють. Для цього необхідне рівномірне розподілення локального навантаження на всьому хмарному середовищі. Завдяки цьому, не тільки збільшується кількість користувачів та рівномірне розподілення навантаження по системі, а також ефективне та економічно вигідне використання ресурсів хмарного середовища.

Для плавного розвитку та забезпечення технологічної підтримки ресурсам серверу, їх збільшення, важливу роль відіграє масштабування. Щоб досягати постійного збільшення ресурсів, найкращим способом є масштабування.

Одним із актуальних способів застосування нових серверів, є автоматичне масштабування. Завдяки даній функції, користувач може задати правила і умови для автоматичного збільшення чи зменьшення виділених обчислювальних потужностей на основі поточного навантаження на додатки [12]. Система буде самостійно збільшувати або зменшувати екземпляри веб-сайтів, хмарних сервісів або віртуальних машин в залежності від заданих умов і поточного навантаження на хмарний додаток. Таким чином, користувачі хмарної платформи зможуть використовувати автоматичне масштабування для швидкого виділення хмарних потужностей при зростанні навантаження і економії коштів, коли навантаження падає.

Властивості ефективного балансування навантаження:

- завантаження системних ресурсів здійснюється рівномірно;
- при будь-якому збільшенні навантаження, алгоритм масштабування продовжує ефективно працювати;

- для чіткого вирішення поставлених задач, необхідно передбачати, за яких саме навантаженнях, алгоритм масштабування буде діяти більш ефективно.

Для досягнення продуктивності паралельних систем та швидкості, балансування навантаження має безпосереднє відношення.

Основними причинами виникнення проблемного балансування навантаження на додатки є:

- різнорідність структури розподілення додатку, оскільки різні обчислювальні потужності потребують різних логічних процесів;
- різнорідність структури обчислювального комплексу: різна продуктивність різних обчислювальних вузлів;
- різнорідність структури взаємодії між вузлами: різна пропускна спроможність ліній зв’язку, які з’єднують вузли.

Мінімізація потреб, які виникають між процесами зв'язку з оптимальним використанням ресурсів та часом роботи є основною метою. Тобто перерозподіл за допомогою завдань збалансованого навантаження.

Основні ознаки алгоритму балансування навантаження:

- Продуктивність. Можливість оптимально збільшити ефективність системи.
- Однаковість роботи. Незалежно від походження або положення всі робочі місця в системі можна обробляти однаково.
- Стійкість. Навіть при наявності часткової відмови системи, зберігається стійкість роботи.
- Видозмінювання. Можливість видозмінюватися в залежності від змін а також розширення конфігурації розподіленої системи.
- Системна незмінність. Навіть при збільшенні або зменшенні навантаження, можливість обліку залишається незмінною.

За допомогою програмних та апаратних інструментів відбувається балансування навантаження. Використання динамічного балансування, з

урахуванням повсякденного завантаження серверу, надає можливість оптимально використовувати наявні ресурси хмарних середовищ.

Для того, щоб досягти баланс навантаження в хмарному середовищі, необхідно розподілити завдання серед вузлів. Наступний крок, це відстеження віртуальної машини та втілення операції балансування навантаження. Здійснюється як за допомогою міграції завдань, так і за підходом міграції віртуальної машини.

Виконання завдань за мінімальний проміжок часу у певний період часу віртуальною машиною за графіком, є основним завданням планування. Це пов’язано з тим, що кількісні показники та розмір завдань в хмарному середовищі дуже швидко змінюються, що ускладнює підрахунок всіх можливих картографічних завдань. Для того, щоб зменьшити кількість перезавантаження/завантаження віртуальної машини, потрібно планувати виконання завдання, яке може ефективно поширюватись.

Для збереження рівноваги у системі, необхідно щоб завдання розподілялись від перевантаженої віртуальної машини, до завантаженої. Це досягається тоді, коли виконується операція планувальником задач з розподілу навантаження.

Для ефективного планування завдавання в ХС необхідно 3 стадії:

1. Користувач. Вимога на обслуговування в частці якості обслуговування (QOS, QUALITY OF SERVICE), апаратного забезпечення, програмного забезпечення. Такі вимоги подаються через графічний або веб-інтерфейси користувача.
2. Планувальник фазових завдань. На цій стадії здійснюється виконання завдань планування та операції з балансування навантаження. Автентичний запит до планувальника завдань для подальшої обробки переадресовується обробником запиту на роботу. Тут формуються відповіді всім завданням віртуальної машини.

Інформація про зайняті та вільні вузли завжди наявна у планувальника.

3. Стадія формування базового планування завдань на рівні хмарні середовища. В центрі з обробки даних містяться хости, які в свою чергу віщують віртуальну машину. В залежності від потужності хосту і кількості запитів від користувачів, кількість віртуальних машин може змінюватись.

Висновки до розділу 1

Проведено детальний огляду сучасних технологій хмарних обчислень, який показав, що враховуючи особливості організації хмарних інфраструктур, споживачі хмарних послуг мають можливість широко використовувати розподілені ресурси. Це дозволяє зменшити витрати на обслуговування та значно розширити діапазон послуг.

Проаналізовано принципи роботи хмарного середовища та ряд моделей обслуговування хмарних обчислень, а сама: SAAS (послуга - програмного забезпечення), PAAS (послуга - платформа), IAAS (послуга - інфраструктура). Розглянуто переваги та недоліки моделей та використання хмарної інфраструктури в цілому.

Розглянуті принципи організації роботи хмарного середовища показали, що на сьогодні залишаються невирішеними питання більш ефективного використання наявних послуг та розподілених ресурсів, пов‘язані з проблемами забезпечення безпеки та конфіденційності при обробці і зберіганні інформації, а також підвищення ефективності обслуговування за рахунок удосконалення розподілу навантаження на сервіси хмарної інфраструктури.

2. ПОРІВНЯННЯ ПОШИРЕНИХ МЕТОДІВ І ПОСЛІДОВНОСТІ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ

2.1. Балансування навантаження – основні визначення

Планування продуктивності є основним завданням під час проектування інформаційних систем. Відповідь на спрямовані запити та швидка реакція є основним критерієм якості. Будь-який ресурс має свою ціну, тому це легше досягти завдяки потужному обчислювальному обладнанню. Тому, наразі, є потреба досягти співрозмірності між навантаженням на проектовану інформаційну систему, яку ми очікуємо, та рекомендованої апаратної частини [1]. Правильний пошук потенційного рівня продуктивності навантаження досягається у соцмережах, Інтернет-магазинах, оголошеннях відео-хостингу і т.д. Даючи рекламу, навантаження на послугу може кардинально зрости в декілька раз. Для комерції, це є позитивним аспектом, проте не слід забувати, що у тому разі, якщо навантаження на сайт перевищує норму, він вийде із ладу. Це не тільки зупинить роботу сайту, але й призведе до збитків, витрачених на саму рекламу, яка більше не надасть бажаного результату. Однак бувають ситуації, коли необхідно змінити продуктивність інформаційної системи у бік збільшення, або зменшення. Надмірне збільшення продуктивності може негативно позначитися на роботі системи. Тому слід враховувати такі ситуації, і вчасно масштабувати, поступово змінюючи продуктивність, не порушуючи при цьому фундаментальну архітектуру системи [3].

2.2 Цілі та правила до балансування системи

Для поділу навантаження між серверами, необхідне балансування. Шляхом балансування можливо не тільки досягти безпеки від хакерських атак та інших уражень, але й підвищити стійкість від відмов нашої системи, тобто при виході одного із серверів з ладу, його роботу бере на себе інший [7].

Як і до будь-якої системи, до балансування висувається дотримання певних правил [13]. Основними цілями є:

- розподільча, тобто рівномірне розподілення навантаження між серверами;
- зменшення відмов системи;
- забезпечення безпеки від хакерських атак.

Правила, які застосовуються до балансування:

- справедливість;
- ефективність;
- час виконання запиту та відгуку повинен бути зменшений;
- очікуваність;
- рівномірність завантаження системних ресурсів;
- масштабування.

Можливість оцінити запропонований метод динамічного балансування навантаження в ХС є основною метою цього дослідження. Окрім цього, встановлення такого розподілу завдань, який зможе забезпечити рівномірне обчислювальне навантаження при мінімальних витратах. Під час виконання програми, перерозподіл обчислювального навантаження досягається динамічним балансуванням [4].

Реалізація динамічного навантаження (Рис.2.1) досягається завдяки програмному забезпеченню, до якого ставляться певні вимоги:

- загрузка вузлів;
- спроможність пропускати лінії зв'язку;
- частота обмінів повідомленнями між процесом розподіленого додатка.

Однією з важливих задач є технологія управління розподіленими ресурсами. В умовах швидкого росту навантаження та збільшення кількості компонентів мереж, забезпечується керування інформаційними інфраструктурами.

Рисунок 2.1 – Реалізація динамічного навантаження

2.3 Способи балансування навантаження

Вирішити питання збільшення балансування навантаження можливо декількома способами. По-перше, це об'єднання кількох однорідних елементів (серверів), яке може розглядатися як самостійна одиниця, що володіє певними властивостями. По-друге, збільшення продуктивності наявного сервера. Обидва методи збільшення навантаження є дієвими, однак у разі поломки серверу, другий спосіб не допоможе [8]. Кластер в даному випадку більш надійний, так як у разі поломки одного із серверів, навантаження перерозподіляється на інші, робочі. Балансувальник самостійно приймає рішення щодо розподілення навантаження між однорідними елементами, при цьому обмежень щодо масштабування немає.

Балансування навантаження ділиться на два види: динамічне та статичне. На стадії проектування розподіленого додатка здійснюється статичне балансування [17]. Розподіл логічних процесів на процесори не є ефективним, хоча дуже часто застосовуються генетичні алгоритми та використовується досвід попередніх запусків.

Вирівнювання навантаження або його балансування є одним із методів розподілу завдань між декількома серверами. Основною метою є

оптимізація використання ресурсів, скорочення часу обслуговування запитів.

2.3.1 Статистичне вирівнювання навантаження

Статистичне балансування розподіляє трафік між серверами послідовно, а для цього необхідно знати системні ресурси. Отже поточний стан системи не має значення к рішенню.

Динамічне балансування, яке приймає рішення, ґрунтується на фактичному стані системи. Це дозволяє в режимі реального часу переміщувати з перевантаженої машини на вільну [3]. Симетричний розподіл заданого обчислювального завдання та зниження вартості зв'язку розподілених обчислювальних вузлів фокусується на алгоритмі змішаного завантаження балансу.

В динамічне середовище, хмарні обчислення потрапляють для того, аби зосередитись на алгоритмі динамічного навантаження. Його також можна класифікувати на періодичне планування пакетного режиму та планування негайного режиму. В першому випадку після збору прибуття завдання призначається відповідним ресурсам [2]. А в другому завдання ресурсом призначається одразу. При цьому важливим є мінімальний час завершення та виконання.

Характеристики, на які впливає алгоритм планування:

- очікування відбувається в мінімальний час;
- швидкий відгук;
- пропускна спроможність на максимальному рівні;
- найбільша завантаженість процесора.

2.3.2 Динамічне вирівнювання навантаження

Динамічне вирівнювання під час виконання програми, розподіляє обчислювальне навантаження на вузли.

При реалізації динамічного вирівнювання в програмному забезпеченні враховується:

- завантаження обчислювальних вузлів;
- пропускна спроможність ліній зв'язку;
- частота обмінів повідомленнями.

За допомогою апаратних і програмних інструментів здійснюється балансування навантаження. Для побудови хмарного середовища, яке оптимально використовує всі наявні ресурси, використовується динамічне балансування, при цьому враховується поточне завантаження серверів [9].

Рішення щодо балансування навантаження, тобто попередні знання про дії в роботі або глобальному стані системи, не передбачені алгоритмом динамічного навантаження, оскільки ґрунтуються лише на існуючому або поточному стані системи.

Алгоритм динамічного навантаження, у розподіленій системи, здійснюється всіма вузлами, які присутні в системі. При цьому завдання балансування навантаження розподіляється між ними. Взаємодія між вузлами для вирівнювання навантаження може здійснюватися у двох формах: кооперативній та не кооперативній [2].

Для кооперативної форми характерна робота вузлів поруч один за одним. Це робиться для просування загального часу відгуку. Для не кооперативної форми вузли працюють незалежно в напрямку локальної мети. Наприклад, для просування часу відповіді локального завдання. Оскільки кожен з вузлів у системі повинен взаємодіяти з кожним іншим вузлом, алгоритм динамічного розподілу навантаження, що має розподілений характер, часто надсилає більше повідомлень, аніж нерозподілені. У разі, якщо один або декілька вузлів не активуються, зупинка процесу вирівнювання навантаження не відбудеться, що є значною перевагою. Це впливає на продуктивність самої системи.

Для нерозподіленого типу характерно, коли один або декілька вузлів виконують задачу вирівнювання навантаження.

Динамічний розподіл навантаження нерозподіленого характеру може мати дві форми, а саме: централізовану та напів-розподілену. У централізованому алгоритмі вирівнювання навантаження здійснюється лише через центральний вузол, який відповідає за вирівнювання навантаження у всій системі, а інші з ним взаємодіють. У напіврозподіленій формі, де вирівнювання навантаження в кожному вузлі (кластер) має централізовану форму. Балансування навантаження всієї системи здійснюється через центральні вузли кожного кластера, тобто у кожному кластері вибирається центральний вузол за допомогою відповідної методики вибору. Це забезпечує вирівнювання навантаження всередині цього кластера [7].

Кількість загальних взаємодій у системі різко зменшується в порівнянні з напів-розподіленою формою, так як централізоване динамічне навантаження вимагає меншої кількості повідомлень для прийняття рішення.

Однак коли центральний вузол виходить з ладу, процес вирівнювання навантаження виявляється не дієвим, а тому підходить для невеликих мереж.

2.4 Процес розподілення вирівнювання навантаження

Попередні знання про дії в роботі або глобальному стані системи не передбачені алгоритмом динамічного навантаження. Рішення щодо вирівнювання навантаження базується лише на існуючому або поточному стані системи. Задачі вирівнювання навантаження розподіляються між ними, оскільки алгоритм динамічного навантаження у розподіленій системі виконується всіма присутніми у системі вузлами. Виділяють дві форми взаємодії для виконання вирівнювання навантаження: кооперативна та некооперативна [12].

Алгоритм кругового обслуговування (Round Robin), представляє алгоритм розподілу навантаження розподіленої обчислювальної системи

методом перебору і впорядкування її елементів по круговому циклу. Спочатку запит передається одному серверу, потім іншому аж до досягнення останнього сервера (Рис.2.2). Далі все починається по колу, тобто процеси розподілені між всіма процесорами. Найпоширенішим алгоритмом розподілення навантаження є Round Robin DNS. Проте не дивлячись на те, що робоче навантаження між процесорами однакове, час обробки завдання для усіх процесів різний. Тому в певний проміжок часу одні вузли можуть бути завантажені, в той час як інші навпаки. Для завантаження запитів балансу між кількома веб-серверами, доцільно користуватись саме Round Robin.

Головні ознаками ефективної роботи є:

- пропускна спроможність;
- невеликий час відгуку;
- рівномірне використання ресурсів;
- продуктивність роботи.

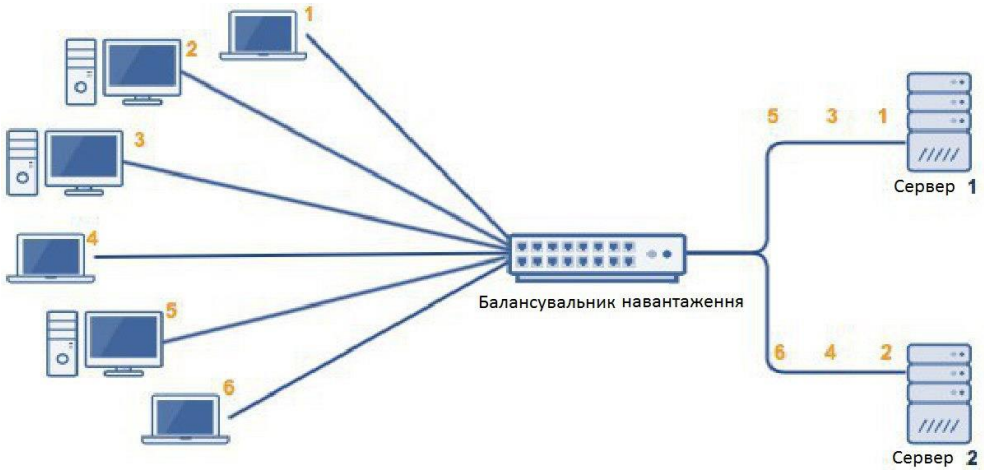

Рисунок 2.2 – Схема мережі Round Robin

Max-min – ще один алгоритм вирівнювання навантаження. Він ідентичний процедурі алгоритму Min-min. Цей алгоритм досягається шляхом розподілу тоді і тільки тоді, коли розподіл можливий, а спроба збільшити розподіл будь-якого учасника обов'язково призводить до

зменшення розподілу іншого учасника з рівним або меншим розподілом. Таким чином обчислюється час завершення виконання всіх завдань. Даний алгоритм краще за все використовувати при максимальному часі завершення завдань. Доки не буде закінчено інше завдання максимального часу завершення, мінімальний час завершення завдання чекатиме у замовленій черзі. Вказаний алгоритм добре працює в статистиці.

Обидва алгоритми мають свої переваги та недоліки на основі середовища. Проте у зв’язку із їх ідентичністю, від обраного алгоритму продуктивність не залежить, вони однаково виконуються у хмарному середовищі.

Connection Mechanism або алгоритм вирівнювання навантаження. Він базується на найменшому механізмі зв'язку, тобто компоненті алгоритму динамічного планування. Для визначення примірного навантаження, необхідно підрахувати кількість підключень кожного із серверів. Connection Mechanism відстежує номер підключення для кожного із серверів. Коли надсилається нове з'єднання, кількість посилань додається. Коли закінчується з'єднання або відбувається тайм-аут - зменшується. Головними показниками є: пропускна спроможність, стійкість до відмов, використання ресурсів, масштабування.

A Task Scheduling Algorithm Based on Load Balancing або алгоритм планування задач. Він ґрунтується на вирівнюванні навантаження. Дворівневий метод планування здійснення завдання ґрунтується на вирівнюванні навантаження. Це здійснюється для сприйняття динамічних вимог користувачів і отримання якісного використання ресурсів. Цим забезпечується вирівнювання навантаження за допомогою першого планування завдань для віртуальних машин, зберігається розміщення ресурсів, збільшується час не тільки відповіді ресурсу, а й споживання ресурсів, загальної продуктивності середовища обчислень у хмарі [11].

Min-min – алгоритм вирівнювання мінімальної місткості. Це дуже простий статичний алгоритм, який показав відмінну продуктивність у

плануванні завдань. Менеджер служби знаходить у хмарі час закінчення кожного завдання, при цьому нове завдання чекає своєї черги для виконання. Такий алгоритм знаходить завдання, яким потрібен мінімальний час виконання для завершення.

Алгоритм Randomized має тип статичного характеру. У цьому алгоритмі процес можна обробляти певним вузлом n з ймовірністю p. Незалежно від розподілу з віддаленого процесора, порядок розподілу процесу зберігається для кожного процесора, у зв’язку з чим забезпечується процес рівномірного завантаження. Однак коли навантаження має різні обчислювальні складності, виникають складнощі. Алгоритм Randomized не підтримує детерміністичний підхід.

Active Clustering – це кластерний алгоритм. Він вводить в хмарне обчислення таке поняття, як кластеризація. Існує багато алгоритмів вирівнювання навантаження у хмарних обчисленнях, які мають свої недоліки та переваги перед іншими. І вже в залежності від цілей, використовують один із таких алгоритмів. Шляхом створення кластера вузлів, який вважається групою, можна покращити продуктивність алгоритму. Принципом Active Clustering є об'єднання подібних вузлів і робота над групами. Завдяки високій доступності ресурсів, підвищується продуктивність системи, а отже і збільшується пропускна здатність. Таке збільшення пояснюється ефективним використанням ресурсів, а саме: часом міграції, витратами, використанням ресурсів.

2.5 Системи вирівнювання навантаження

HAProxy – це серверне програмне забезпечення, яким забезпечується висока доступність та вирівнювання навантаження для TCP і HTTP-додатків, за допомогою розподілу вхідних запитів на кілька обслуговуючих серверів (Рис.2.3). HAProxy – пропонує високу доступність, вирівнювання навантаження та проксі-сервер для програм. Сьогодні, HAProxy - це стандартний балансувальник відкритого ресурсу, з

дуже високим рівнем трафіку, такого як, наприклад, веб-сайти. Не зважаючи на безкоштовність, він швидкий та досить надійний.

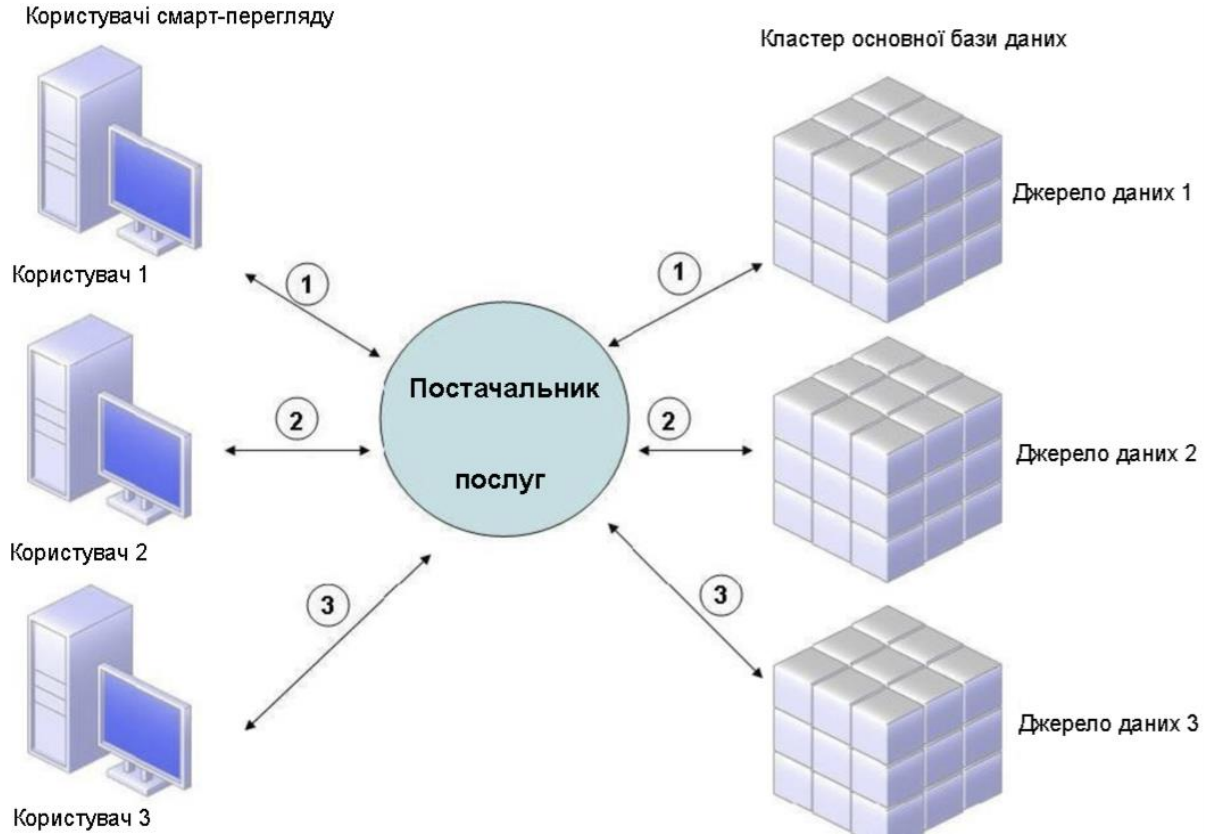

Рисунок 2.3 – Схема роботи HAProxy

Режим роботи такої системи легкий та без ризиків, проте у не стійких веб-сервісах мережі Інтернет краще не використовувати [7].

HAProxy доступно декілька версій. Основною відмінністю між ними є набір функцій. Основними з них для кожної з версій є:

- версія 1.8 – має багато потоків, HTTP/2, кеш-пам'ять, можливість додавання/видалення серверу, безперервне перезавантаження, dns srv, апаратні SSL-двигуни і т.д.;
- версія 1.7 – додана серверна швидка реконфігурація, містяться агенти обробки вмісту, мультиструктурні сертифікати і т.д.;
- версія 1.6 – можливість підтримувати роздільну спроможність dns, мультиплексування HTTP - з’єднання, наявність повної реплікації стільникового з’єднання, стиснення безготівкових даних і т.д.

- версія 1.6 – додані SSL, IPv6, захист DDoS тощо. Кожна версія визначає свій набір функцій, які оновлюють попередню.

Покращення сумісності є важливим аспектом HAProxy, оскільки навіть версія 1.5 може працювати з конфігураціями, зробленими для версії 1.0 тринадцять років тому.

Задля досягнення абсолютно максимальної продуктивності, HAProxy виділяє декілька методів, які зазвичай зустрічаються в архітектурі ОС:

- модель, яка керується подіями. Вона значно зменшує вартість контекстного перемикача та використання пам'яті. Можливість обробки сотні завдань за одну мілісекунду, при цьому використання пам'яті становить лише декілька кілобайт за сеанс, в той час як споживана пам'ять на попередньо встановлених або жорстких серверах більше декількох мегабайтів на процес;
- перевірка подій на системах, які це підтримують (Linux і FreeBSD). Це надає змогу миттєво виявляти будь-яку подію серед десятків тисяч, в будь-якому з'єднанні;
- затримка оновлень перевірки подій за рахунок «ледачого» кеша подій, що дає змогу заощадити багато системних викликів;
- буферизація без будь-якої копії даних між читанням та записом. Це дає змогу заощадити багато циклів процесора та корисної пропускної спроможності пам'яті;
- перенаправлення нульової копії, яка можлива за допомогою системного виклику splice() під Linux, в результаті якого здійснюється реальна нульова копія, починаючи з Linux 3.5. Це дозволяє невеликому пристрою під 3 Вт, пересилання HTTP-трафіку на один Гбіт/с;
- розподілювач пам'яті MRU з використанням пам'яті з фіксованим розміром, що віддає перевагу ділянкам «гарячої» кеш-пам'яті над «холодними» кешами. Це значно скорочує час, необхідний для створення нового сеансу;

- здатність до обмеження кількості accept() для ітерації при роботі в багатопроцесному режимі, так що навантаження рівномірно розподіляється між процесами;
- адаптація до апаратного забезпечення та максимально можливого ядра процесора, який керує мережевими адаптерами, але не суперечить йому;
- оптимізована черга таймера;
- оптимізований аналіз заголовків HTTP: заголовки аналізуються за допомогою інтерпретації, також має оптимізований синтаксичний аналіз.

Це дає змогу уникнути повторного читання будь-якої раніше прочитаної області пам'яті. Контрольна точка використовується, коли кінцевий буфер досягається з неповним заголовком, так що синтаксичний аналіз не починається з початку, коли більше даних будуть прочитаними. Розбір середнього HTTP-запиту зазвичай займає півмісяця на швидкому Xeon E5:

- ретельне зменшення кількості дорогих системних дзвінків. Більша частина роботи виконується в користувацькому просторі за замовчуванням, наприклад, читання часу, агрегування буферу, включення та вимкнення файлового дескриптора;
- контент-аналіз оптимізовано для того, щоб носити лише покажчики на оригінальні дані та ніколи не копіювати, якщо дані не потрібно трансформувати. Це гарантує те, що дуже малі структури переносяться і що вміст ніколи не реплікується, коли це не є необхідним.

Пропозиція Cloud Computing варіює від пропозиції конкретної IT-інфраструктури до розгортання складних програм та програмних рішень [11]. Вивчаючи модель доставки послуг Cloud Computing, виникає завдання управляти сотнями тисяч запитів користувачів і програм. Тому постачальник Cloud Computing повинен розглянути розумну інтеграцію інфраструктури, щоб створити пропозицію Cloud Computing, яка забезпечує прозорість, масштабованість, безпеку та найвищу якість (QoS).

Для постачальників Cloud, які планують надавати певні послуги та користувачів, оцінка хмарних обчислень є обов'язковою. Особливо, якщо вони мають намір перенести свою інформаційно-технічну інфраструктуру, платформу або програмне забезпечення в Cloud [12]. Незважаючи на те, що використання реальної інфраструктури для тестування та оцінки розгортання хмар може дати дослідникам реальний підхід до прийняття критичного рішення про рух вперед з цією моделлю обчислень, в більшості випадків це може бути дуже дорогим:

- інфраструктури, платформи та програмне забезпечення - це достатньо великі витрати;
- необхідність тестування на масштабованих середовищах (більше інфраструктури);
- витрати на управління та технічне обслуговування. Окрім цього критичного чинника, можна додати споживання часу, щоб протестувати конкретний сценарій:
- інфраструктура встановлення та конфігурації;
- повторювані та змінні тести;
- налагодження та усунення несправностей.

Використання інструментів моделювання є більш життєздатною альтернативою. Вони дозволяють оцінити дослідження порівняльного аналізу застосування в контрольованому середовищі, де можливо легко відтворити результати.

Для моделювання великомасштабних середовищ хмарного обчислення, існує безліч інструментів стимулятора.

Можна визначити два типи симуляторів: графічний інтерфейс користувача (GUI) або симулятори на основі мови програмування (наприклад, Java).

Як описано в попередньому розділі, існує різноманітна кількість симуляторів Cloud Computing, кожна з яких має специфічні характеристики та орієнтована на конкретну ціль. Вибір найкращого

симулятора Cloud Computing - складна місія [11]. Наскільки нам відомо, було важко визначити CloudSim, як основну платформу для найбільш часто використаних симуляторів Cloud до цього моменту. CloudSim був створений, як розширення симулятора GridSim, щоб змоделювати шар віртуалізації Cloud Computing, який не був присутній в оригінальному симуляторі [11].

Придатною альтернативою є використання інструментів моделювання. Тоді можливо оцінити гіпотезу перед розробкою програмного забезпечення в середовищі, де можна відтворити тести. Наприклад, у випадку обласного обчислення, коли доступ до інфраструктури здійснює платежі в реальній валюті, підходи на основі моделювання дають значні переваги, оскільки це дозволяє Cloud клієнтам протестувати свої послуги в повторюваному та контрольованому середовищі безкоштовно, а також налаштовувати продуктивність вузьких місць до розгортання на реальних хмарах. На стороні постачальника, середовища симуляції дозволяють оцінити різні види сценаріїв довгострокової оренди ресурсів при різному розподілі навантаження та ціноутворення. Такі дослідження могли б допомогти постачальникам оптимізувати вартість доступу до ресурсів з наголосом на підвищення прибутку.

CloudSim - це симулятор, заснований на мові програмування, і навіть якщо він не підтримує графічний інтерфейс користувача для моделювання, він пропонує CloudAnalyst (який є розширенням CloudSim) для дослідників, які вважають за краще використовувати зручний інтерфейс для проведення своїх досліджень (Рис.2.4). CloudSim позиціонує себе дослідникам хмарних обчислень як Java-платформу, яка підтримує основні характеристики Cloud Computing (IaaS) з підтримкою віртуалізації та плануванням завдань (PaaS і SaaS) і відкриває двері для виникнення, інтеграції та тестування нових алгоритмів для планування завдань або розробки нових характеристик, що допомогло в доставці нових тренажерів.

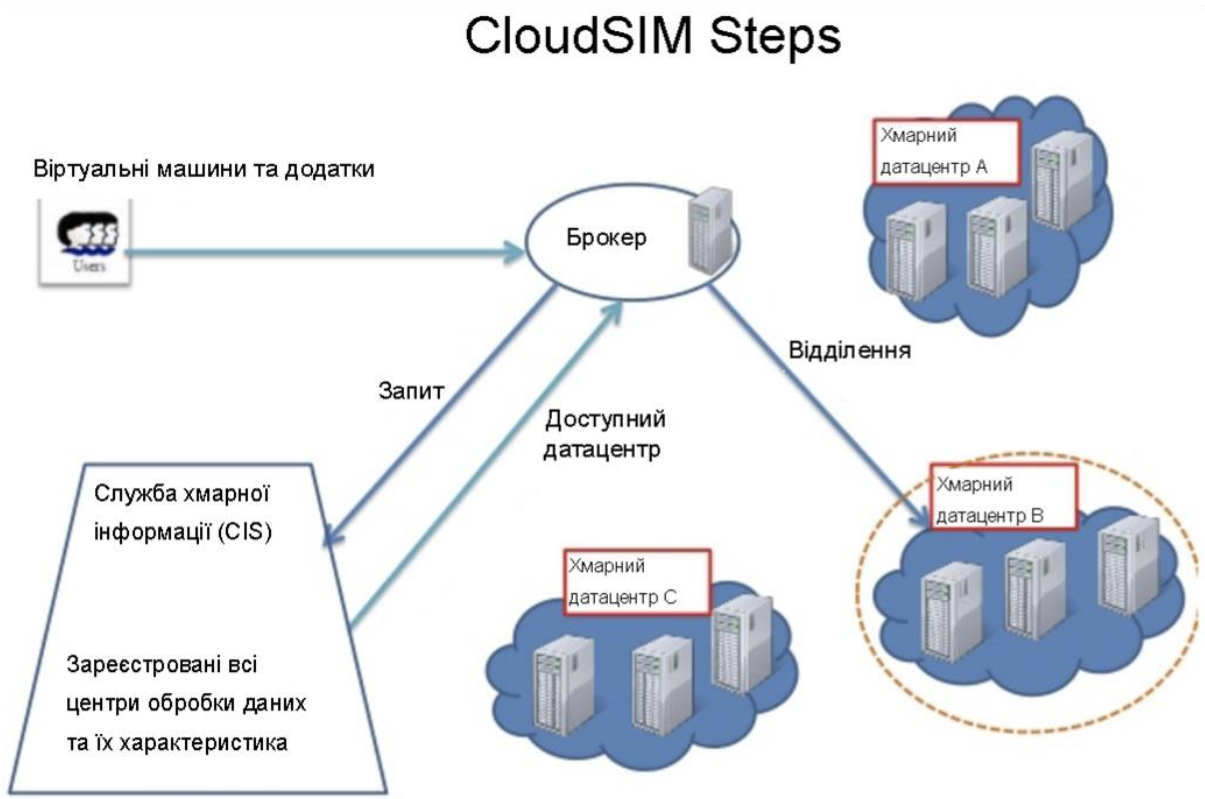

Рисунок 2.4 – Схема роботи симулятора CloudSim

Основні функції CloudSim:

- підтримує моделювання великих центрів обробки даних ХО;
- підтримує та моделює віртуалізовані серверні хости з налаштованою політикою, для забезпечення ресурсів хоста віртуальних машин;
- підтримання та моделювання контейнерів для додатків;
- підтримка та моделювання енергозберігаючих обчислювальних ресурсів;
- підтримка та моделювання мережевих топологій центрів обробки даних та програм передачі повідомлень;
- підтримка та моделювання федеративних хмарних середовищ;
- підтримка динамічної вставки елементів симуляції, зупинки та відновлення симуляції;
- підтримка визначеної користувачем політики для виділення хостів на віртуальні машини та політики для розподілу ресурсів хоста на віртуальні машини.

Висновки до розділу 2

Розглянуто основні правила та способи балансування навантаження, які обумовлюють процес розподілення та вирівнювання навантаження, не порушуючи при цьому фундаментальну архітектуру хмарного середовища.

Проведено методологічний аналіз існуючих систем вирівнювання навантаження та обґрунтовано використання кожної з них для оптимальної обробки запитів різних користувачів, враховуючи показники ефективності систем та переваги і недоліки при виконанні певних задач. Такий аналіз показав, що питання вирівнювання навантаження «хмари» потребує удосконалення процесу розподілу робочого навантаження для підвищення ефективності використання ресурсів та зменшення часу обробки запитів.

3. РОЗРОБКА СИСТЕМИ НА ОСНОВІ МОДИФІКОВАНОГО СПОСОБУ ДИНАМІЧНОГО БАЛАНСУВАННЯ НАВАТАЖЕННЯ

3.1 Використанні технології

3.1.1 Мова програмування Python

Основною мовою розробки, для створення системи балансування навантаження у хмарному середовищі було обрано Python версії 3.6.3. Дана мова програмування має свої переваги і недоліки [19]. Серед основних переваг мови програмування Python можна навести такі:

- Мультиплатформенність, завдяки цьому систему можна буде використовувати на будь якій операційній системі, головною вимогою буде встановлений python 3, а також набір сторонніх бібліотек які описані в файлі requirements.txt.
- Легка масштабованість. це означає що можливе масштабування систему у різні боки, як до збільшення, так навпаки до зменшення [21].
- Мультипарадигменість. Мова підтримує 3 основні парадигми програмування. Об'єктно-орієнтованість, функнкціональність і процедурність, це дозволяє розширити інструменти роботи, і полегшує розробку системи.
- Підтримка великої кількості бібліотек, що дозволяють збільшити швидкість розробки системи, а також полегшує її розробку [19].

Але також є недоліки, серед них можна виділити:

- Низьку швидкість роботи, адже мова програмування Python 3 є інтерпретованої з динамічною типізацією, це означає що деякий час виконання роботи, система витрачає на з'ясування типу даних змінних [22].
- Погана робота з потоками, так як у мові реалізований блокувальник GIL, що не дозволяє виконуватися двум чи більше потокам одночастно.

Проаналізувавши всі недоліки і переваги даної мови програмування, було вирішено обрати його для розробки системи балансування навантаження в хмарному середовищі.

3.1.2 Менеджер пакетів pip

Менеджер пакетів або система керування пакетами – це набір програмних засобів, що автоматизує процес встановлення, модернізації, налаштування та видалення комп'ютерних програм для операційної системи комп'ютера послідовним методом.

Менеджер пакетів займається розподілом програмного забезпечення та даними у файлах архіву. Пакети містять метадані, такі як ім'я програмного забезпечення, опис його призначення, номер версії, постачальник, контрольна сума та список залежностей, необхідних для правильного запуску програмного забезпечення. Після встановлення метадані зберігаються в локальній базі даних пакетів. Менеджери пакетів, як правило, підтримують базу даних про програмні залежності та інформацію про версію для запобігання несумісності програмного забезпечення та відсутності передумов. Вони тісно співпрацюють з програмними сховищами, адміністраторами бінарних репозиторіїв та магазинами додатків.

Пакетні менеджери призначені для усунення необхідності вручну встановлювати та оновлювати. Це може бути особливо корисним для великих підприємств, операційні системи яких базуються на Linux та інших системах, подібних до Unix, які зазвичай складаються з сотень або навіть десятків тисяч різних програмних пакетів.

Пакет програмного забезпечення – це архівний файл, що містить комп'ютерну програму, а також необхідні метадані для його розгортання. Комп'ютерна програма може бути в вихідному коді, яка повинна бути скомпільована і побудована спочатку [16]. Пакетні метадані включають в

себе опис пакета, пакетну версію та залежності (інші пакети, які потрібно встановити заздалегідь).

Пакетні менеджери покладаються на завдання пошуку, встановлення, обслуговування або видалення програмних пакетів за командою користувача, що показана на рис. 3.1. Типові функції системи управління пакетами включають:

- Робота з архіваторами файлів для вилучення архівів пакунків;
- Забезпечення цілісності та автентичності пакету шляхом перевірки їхніх цифрових сертифікатів та контрольних сум;
- Шукати, завантажувати, встановлювати або оновлювати існуюче програмне забезпечення з сховища програмного забезпечення або магазину додатків;
- Групування пакетів по функціях, щоб зменшити плутанину користувачів;
- Управління залежності, щоб забезпечити встановлення пакета з усіма необхідними пакунками, уникаючи "dependency hell".

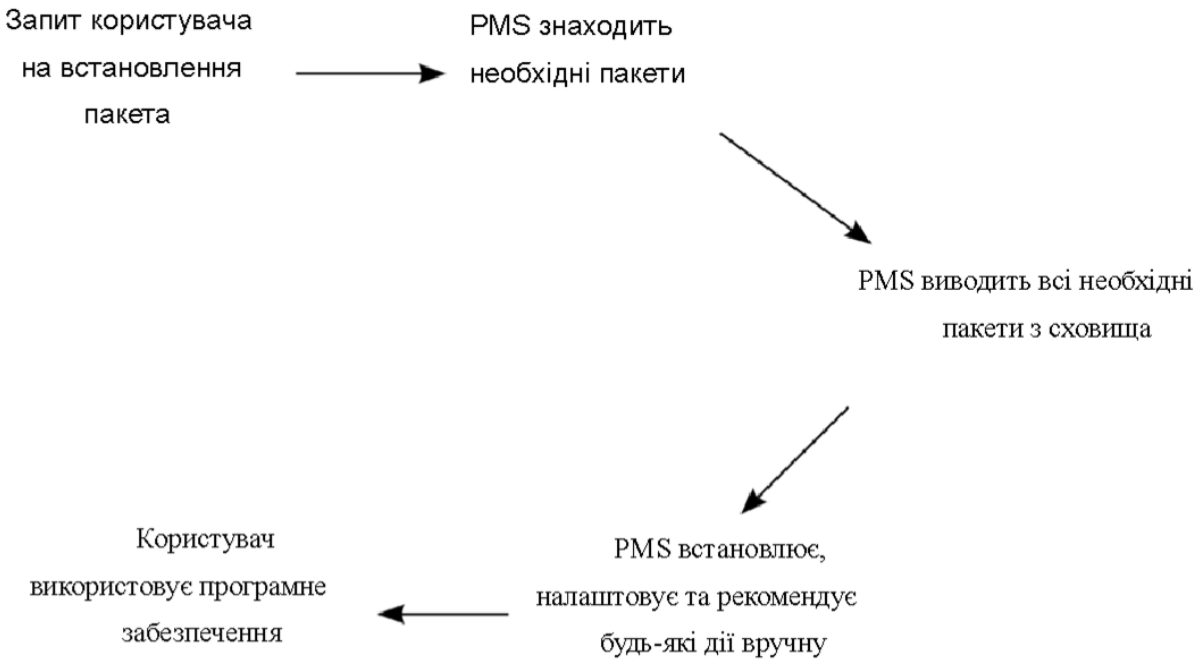

Рисунок 3.1 – Схема встановлення пакетів

Pip (рекурсивний акронім для «Pip Installs Packages») – система управління пакетами, яка використовується для встановлення та керування пакетами програмного забезпечення, написаними на Python. Багато пакетів можна знайти в джерелі за замовчуванням для пакетів та їх залежностей - Python Package Index (PyPI).

Python 2.7.9 і більш пізні версії (в ряд Python2), а Python 3.4 і більш пізні версії включають pip (pip3 для Python 3) за замовчуванням [22].

3.1.3 База даних MongoDB

В якості бази даних було обрано не реляційну базу MongoDB. Є декілька обґрунтувань такого вибору. Перш за все тому, що дана база є не реляційною, отже їй притаманні всі переваги не реляційних баз даних. Не реляційні бази даних були створені для уникнення жосткої структуризації даних, як у їх реляційних аналогах [23]. Це означає, що с самого початку розробки продукту, не потрібно задавати обов'язкову структуру даних, і в процесі розробки програмного забезпечення можливі зміни бази даних. У порівнянні з реляційною базою, вони більш масштабуємі і забезпечує хорошу продуктивність роботи, а така модель даних усуває деякі недоліки [24].

Така модель даних забезпечує можливість обробки:

- Великих об'ємів структурованих, напівструктурованих і неструктурованих даних;
- Об'єктно-орієнтованих даних, які просто представляються і записуються у форматі записів у не реляційну базу даних;
- Ефективна, масштабована архітектура замість монолітної.

Серед переваг такої структури бази даних, можна виділити:

- Відсутність схеми. База даних MongoDB заснована на колекціях різних документів. Проте кількість полей, зміст і розмір цих документів може змінюватись;
- Досить зрозуміла структура кожного об’єкту;

- Адаптована для високонавантажених систем. Для зберігання використаних в даний момент даних використовується внутрішня пам’ять, що дозволяє отримувати швидкий доступ до даних;
- Дані зберігаються у виді JSON-документів;
- MongoDB підтримує динамічні записи документів.

3.1.4 Система обміну повідомленнями RabbitMQ

RabbitMQ – це розподілена система управління чергою повідомлень. Розподілена, оскільки зазвичай працює як кластер вузлів, де черги розподіляються по вузлах як показана на рис.3.2.

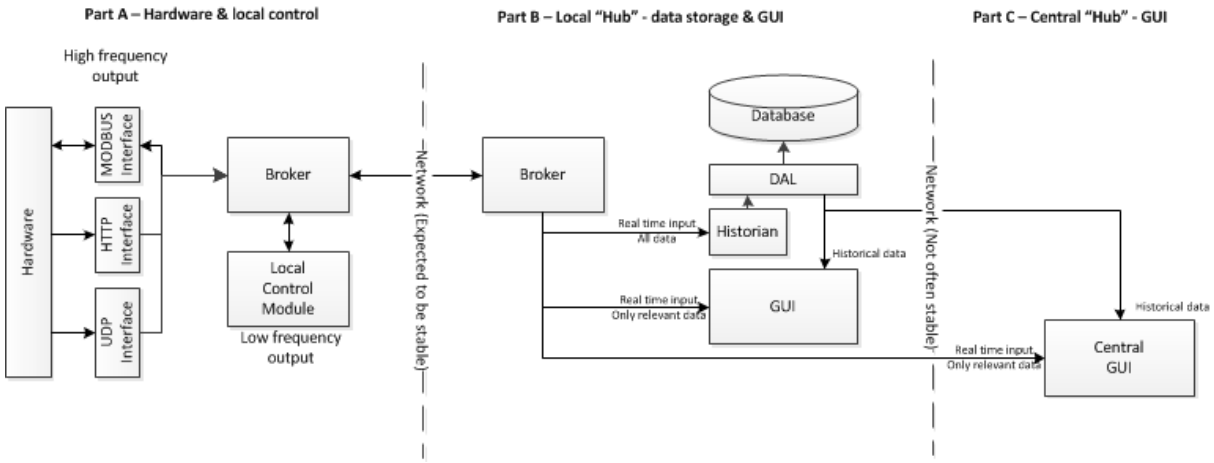

Рисунок 3.2 - Робота розподіленої системи RabbitMQ

Переваги цієї системи:

- Асинхронні повідомлення. Підтримка декількох протоколів обміну повідомленнями, організації черги повідомлень, підтвердження доставки, гнучка маршрутизація до черг, множинний тип обміну [25].
- Розподілене розгортання. Розгортання у вигляді кластерів для забезпечення високої доступності і пропускної здатності.
- Передача повідомлень між брокерами без необхідної кластеризації через декілька зон доступності (регіонів).

- Готові рішення. Підключається аутентифікація, авторизація, підтримка TLS (Transport Layer Security) і LDAP (Lightweight Directory Access Protocol). Легкий і простий у розгортанні в загальнодоступних і приватних хмарах [25].
- Управління і моніторинг. HTTP-API (HyperText Transfer Protocol – Application Programming Interface), інструмент командного рядка і призначений для користувача інтерфейс для управління і моніторингу RabbitMQ.

RabbitMQ використовує як класичний, так і новаторський підходи до обміну повідомленнями. Класичний в тому сенсі, що вона орієнтована на чергу повідомлень, а новаторський – в можливості гнучкої маршрутизації. Саме ця можливість маршрутизації є її унікальною перевагою [25]. Створення швидкої, масштабованої і надійної розподіленої системи повідомлень само по собі є досягненням, але функціональність маршрутизації повідомлень робить її справді визначною серед безлічі технологій обміну повідомленнями.

3.1.5 Google Cloud SDK

Google Cloud SDK - це набір інструментів для Cloud Platform. Він містить gcloud, gsutil і bq, які ви можете використовувати для доступу до Google Compute Engine, Google Cloud Storage, Google BigQuery та іншим продуктам і сервісам з командного рядка [9]. Також можливо запускати ці інструменти в інтерактивному режимі або в автоматизованих сценаріях.

Основними інструментами хмарної платформи є:

- Управління екземплярами віртуальної машини. gcloud спрощує управління вашим парком віртуальних машин на Compute Engine - все, починаючи зі створення, запуску та управління екземплярами віртуальних машин, на згортання ваших власних зображень VM. Також можна використовувати gcloud для підключення SSH доступ до екземплярів [10].

- Мережі, міжмережеві екрани, дисковий сховище. Можна використовувати gcloud для управління мережами Compute Engine, брандмауерами, дисковим сховищем і т. д. Без використання консолі Cloud Platform. З gcloud управління конфігураціями для середовища Compute Engine.
- Запуск локальних сервісних емуляторів. Хмарні емулятори SDK для Google Cloud Pub / Sub і Google Cloud Datastore дозволяють імітувати ці служби у вашому середовищі для локальної розробки, тестування та перевірки. Ви запускаєте і керуєте сервісними емуляторами за допомогою інструменту gcloud.

Можливості Cloud SDK, інтрфейсу командного рядку для Google Cloud Platform.

- Інструмент gcloud, який управляє аутентифікацією, локальною конфігурацією, і взаємодією з API платформи Google Cloud.
- Інструмент gsutil, який надає доступ до командного рядку для управління об’єктами Cloud Storage.
- Інструмент Kubectl, що спрощує запуск і управління кластерами контейнерів.
- Інструмент bq, що дозволяє запускати запити, керувати наборами даних, таблицями і об’єктами BigQuery через командний інтерфейс.

3.1.6 Docker

Docker – це інструмент, призначений для спрощення створення, розгортання і запуску додатків з використанням контейнерів. Контейнери дозволяють розробнику розміщувати додаток з усіма необхідними йому частинами, наприклад бібліотеками та іншими залежностями, і відправляти все це як один файл [26]. Таким чином, завдяки контейнеру, розробник може бути впевнений, що додаток буде працювати на будь-якому іншому комп'ютері Linux незалежно від будь-яких параметрів, що

налаштовуються, які може мати комп'ютери, які можуть відрізнятися від машини, використовуваної для написання і тестування коду.

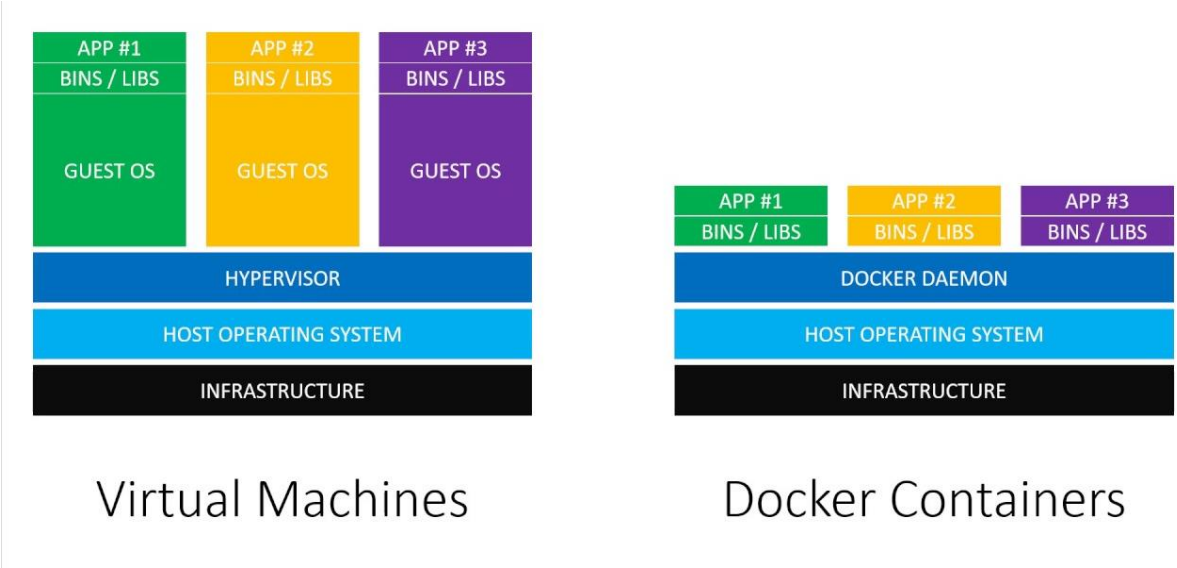

Рисунок 3.3 – Відмінності у структураъ контейнеру і віртуальної машини

У певному сенсі, Docker трохи схожий на віртуальну машину, як це показана на рис. 3.3. Але на відміну від віртуальної машини, а не для створення цілої віртуальної операційної системи, Docker дозволяє додаткам використовувати те ж саме ядро Linux, що і система, в якій вони працюють, і вимагає, щоб програми були з’єднані з програмами, які ще не запущені на головному комп'ютері. Це дає значне підвищення продуктивності і зменшує розмір програми.

І що важливо, Docker з відкритим вихідним кодом. Це означає, що будь-хто може внести вклад в Docker і розширити його, щоб задовольнити свої власні потреби, якщо їм потрібні додаткові функції, які не доступні наразі.

Docker – це інструмент, який призначений як для розробників, так і для системних адміністраторів, що робить його частиною багатьох інструментів DevOps (розробників + операцій). Для розробників це означає, що вони можуть зосередитися на написанні коду, не турбуючись

про систему, в якій вона в кінцевому рахунку буде працювати. Це також дозволяє їм почати роботу, використовуючи одну з тисяч програм, уже розроблених для запуску в контейнері Docker в якості частини їх застосування. Для операційного персоналу Docker дає гнучкість і потенційно зменшує кількість систем, необхідних через його невеликого розміру і менших накладних витрат [26].

Docker має багато переваг:

1. Повернення інвестицій і витрат. Першим перевагою використання Docker є ROI. Найбільшим драйвером більшості управлінських рішень при виборі нового продукту є повернення інвестицій. Чим більше рішення може знизити витрати при збільшенні прибутку, тим краще рішення, особливо для великих, встановлених компаній, які повинні отримувати стійкий дохід в довгостроковій перспективі. У цьому сенсі Docker може допомогти полегшити цей вид заощаджень, різко скоротивши ресурси інфраструктури. Природа Docker полягає в тому, що для запуску одного і того ж додатка потрібно менше ресурсів. Через скорочених вимог до інфраструктури, які має Docker, організації можуть заощадити на всіх, від вартості сервера до співробітників, необхідних для їх підтримки. Docker дозволяє інженерним командам бути менше і ефективніше.

2. Стандартизація і продуктивність. Контейнери Docker забезпечують узгодженість в декількох циклах розробки, випуску та стандартизації вашої середовища. Одним з головних переваг архітектури Docker є стандартизація. Docker забезпечує повторюваність розробки, складання, тестування і виробництва. Стандартизація інфраструктури обслуговування по всьому конвеєру дозволяє кожному члену команди працювати в середовищі паритету виробництва. Роблячи це, інженери більш оснащені для ефективного аналізу та виправлення помилок в додатку. Це зменшує час, що витрачається на дефекти, і збільшує час, доступне для розробки функції.

Як було згадано вище, контейнери Docker дозволяють вам фіксувати зміни в ваших зображеннях Docker і їх контролі версій. Наприклад, якщо ви виконуєте оновлення компонента, яке розбиває всю вашу середу, дуже легко відкотитися до попередньої версії вашого зображення Docker. Весь цей процес можна протестувати через кілька хвилин. Docker працює швидко, що дозволяє швидко виконувати повтори. Крім того, запуск зображень Docker виконується дуже швидко, що можна порівняти з запуском нескладного скрипту. Docker дозволяє створювати образ контейнера і використовувати те ж зображення на кожному етапі процесу розгортання. Величезна перевага цього – можливість відокремити незалежні кроки і запустити їх паралельно. Зокрема, можна прискорити час, що витрачається на складання зображення.

4. Сумісність і ремонтопридатність. Усувається проблема, коли програма працює тільки на машині програміста раз і назавжди. Одним з переваг, які цінує вся команда, є паритет. Парність, з точки зору Docker, означає, що ваші зображення працюють однаково незалежно від того, на якому сервері або на чийому ноутбуці вони працюють. Для ваших розробників це означає менший час, що витрачається на налаштування оточення, налагодження проблем, пов'язаних з конкретним середовищем. Паритет також означає, що ваша виробнича інфраструктура буде більш надійною і зручною в обслуговуванні.

5. Простота і швидкі зміни. Одним з ключових переваг Docker є те, як це спрощує розробку. Користувачі можуть самостійно вибирати конфігурацію, поміщати її в код і розгорнути її без будь-яких проблем. Оскільки Docker можна використовувати в самих різних середовищах, вимоги інфраструктури більше не пов'язані з навколишнім середовищем додатка.

6. Швидке розгортання. Докеру вдається скоротити час розгортання до декількох секунд. Це пов'язано з тим, що він створює контейнер для

кожного процесу і не завантажує ОС. Дані можуть бути створені і знищені, не побоюючись, що витрати на їх відновлення будуть значні.

7. Безперервне розгортання і тестування. Docker забезпечує узгоджену середу від розробки до робочих серверів. Контейнери докери сконфігуровані для підтримки будь-яких змін і залежностей усередині. Таким чином, ви можете використовувати один і той же контейнер від розробки до робочого стану, щоб переконатися, що немає розбіжностей або ручного втручання.

Якщо вам потрібно виконати оновлення під час циклу випуску продукту, ви можете легко внести необхідні зміни в контейнери Docker, протестувати їх і реалізувати ті ж зміни в існуючих контейнерах. Така гнучкість є ще одним ключовим перевагою використання Docker. Docker дійсно дозволяє створювати, тестувати і випускати зображення, які можна розгорнути на декількох серверах. Навіть якщо новий патч безпеки доступний, процес залишається тим самим. Можна застосувати патч, протестувати його і почати використовувати на робочих серверах.

8. Платформи з декількома хмарами. Можливо, це одна з найбільших переваг Докера. За останні кілька років всі великі постачальники хмарних обчислень, включаючи Amazon Web Services (AWS) і Google Compute Platform (GCP), додали індивідуальну підтримку. Контейнери-докери можна запускати всередині примірника Amazon EC2, примірника Google Compute Engine, сервера Rackspace або VirtualBox, за умови, що ОС хоста підтримує Docker.

9. Ізоляція. Docker гарантує, що ваші програми і ресурси будуть ізольовані. Docker гарантує, що кожен контейнер має свої власні ресурси, які ізольовані від інших контейнерів. Можливе використання різних контейнерів для окремих додатків, в яких використовуються абсолютно різні стеки технологій. Docker допомагає забезпечити чисте видалення додатків, оскільки кожне додаток працює на своєму власному контейнері. Якщо вам більше не потрібно додаток, ви можете просто видалити його

контейнер. Він не залишає тимчасових або конфігураційних файлів на вашій ОС хоста.

Крім цих переваг, Docker також гарантує, що кожен додаток використовує тільки ресурси, які їм були призначені. У конкретному додатку не використовуватимуться всі ваші доступні ресурси, що зазвичай призводить до погіршення продуктивності або простому простою для інших додатків.

10. Безпека. І остання перевага використання докерів - це безпека. З точки зору безпеки Docker гарантує, що додатки, що працюють в контейнерах, повністю ізольовані один від одного, надаючи вам повний контроль над потоком трафіку і управлінням. Контейнер Docker не може переглядати процеси, запущені всередині іншого контейнера. З архітектурної точки зору кожен контейнер отримує свій власний набір ресурсів, від обробки до мережевих стеків [26].

3.2 Структура програми

В структурі системи можна виділити чотири основні модулі, що відповідають за різні компоненти системи. До таких основних модулів, можна виділити:

- модуль Env;
- модуль Src;
- модуль Deployment config;
- модуль Project storage.

Далі будуть детально розглянуті кожен з цих модулів, їх призначення та використання, а також їх взаємодія.

3.2.1 Модуль env

Огляд структури програми системи розподілення навантаження, почнемо з модуля під назвою env. Цей модуль відповідає за створення та зберігання python virtual environment. Python як і більшість сучасних мов

програмування має свій унікальний спосіб завантаження, зберігання і розширення пакетів чи модулів. Існує декілька різних локацій, де зберігаються стороні пакети та модулі, що використовуються під час інтерпретації коду на мові python. Для різних систем вони зберігаються в різних системних папках. Для операційної системи Mac OS вони зберігаються в '/System/Library/Frameworks/Python.framework/Versions/x.x', де x.x позначається версія мови python. Для операційної системи Ubuntu Linux, починаючи з версії 12.04 вони зберігаються в системній папці /usr/local/lib/pythonx.x/site-packages, де x.x так само відповідають за версію мови python. За замовчуванням кожен із сторонніх пакетів системи буде встановлений саме у ці директорії операційних систем. Одна в такій системі виникає проблема версійності.

Припустимо що є 2 проекти: progect_A та progect_B, що обидва використовують фреймворк Django. Однак progect_A, працює на версії 2.1.3 а progect_B працює на версії 1.11.13 того самого фреймворку. Отже для запуску цих двух проектів потрібні різні версії одного й того самого фреймворку. Це реальна проблема, так як обидві ці версії фреймворку будуть знаходитися в модулі site-packages, та мати однакове ім'я каталогу, при цих умовах стає неможливим запустити обидва проекти не витрачаючи зайвого часу. Оскільки проекти зберігаються відповідно до їхнього імені, немає різниці між версіями. Таким чином, обидва проекти, project_A та project_B повинні використовувати таку ж версію, що у багатьох випадках є неприйнятним.

В таких випадках потрібно використовувати віртуальне середовище мови програмування python. Основною метою віртуальних середовищ Python є створення ізольованого середовища для проектів python. Це означає, що кожен проект може мати власні залежності, незалежно від того, які залежать від кожного іншого проекту.

Найкраще в цьому полягає в тому, що немає обмежень на кількість середовищ, які ви можете мати, оскільки це просто каталоги, що містять

кілька скриптів. Крім того, їх легко створювати за допомогою інструментів командного рядка virtualenv або pyenv.

Після створення віртуального середовища, для початку роботи його потрібно активувати, для того щоб визначити ключову функцію його використання. Для прикладу на наступних рисунках буде проілюстровано шлях до використовуємого інтерпретатора python. На рисунку 3.4 буде показаний шлях до загального інтерпретатора, тоді коли на рисунку 3.5 буде шлях до інтерпретатору що знаходиться у віртуальному середовищі.

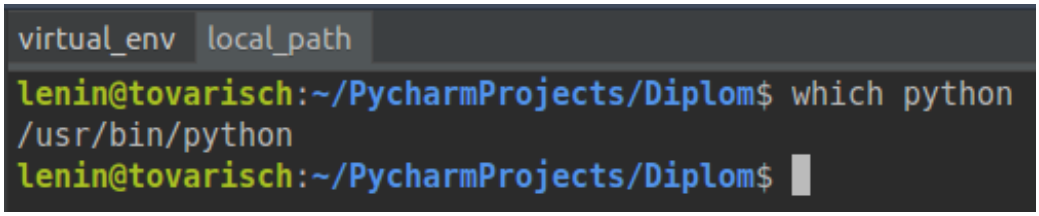

Рисунок 3.4 – Шлях до загального інтерпретатору мови python

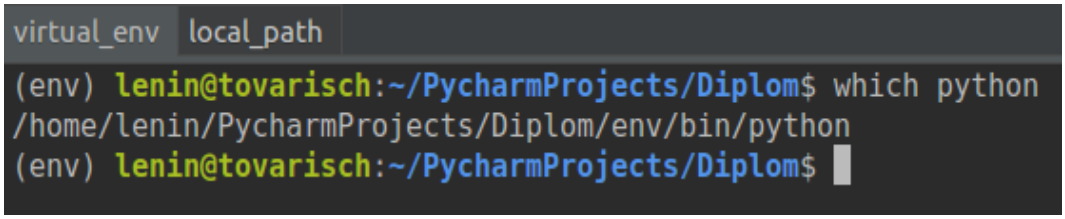

Рисунок 3.5 – Шлях до інтерпретатору мови python, що знаходиться у virtualenv

Це можна пояснити тим, як запускається Python і де він знаходиться в системі. Насправді немає ніякої різниці між цими двома виконуваними файлами Python. Це їх розташування в каталогах має значення.

Коли Python запускається, він дивиться на шлях свого виконуваного файлу. У віртуальному середовищі насправді це всього лише бінарна копія системи Python або її посилання. Потім він встановлює місце розташування sys.prefix і sys.exec_prefix на основі цього місця розташування, опускаючи частину шляху.

Шлях, розташований в sys.prefix, потім використовується для пошуку каталогу site-packages шляхом пошуку відносного шляху lib/pythonx.x/site-packages/, де X.X - це версія використовуваного Python.

Отже модуль під назвою env, зберігає у собі розташування інтерпретатору, для мови програмування python.

3.2.2 Модуль src

Це другий модуль, що використовується у системі, і в ньому зберігається основна частина системи. В загальному розумінні, папка src створюється у проектах заради збереження в ній вихідного коду продукту. В даній магістерській дисертації в ній зберігаються наступні під модулі і файли, що зображені на рисунку 3.6

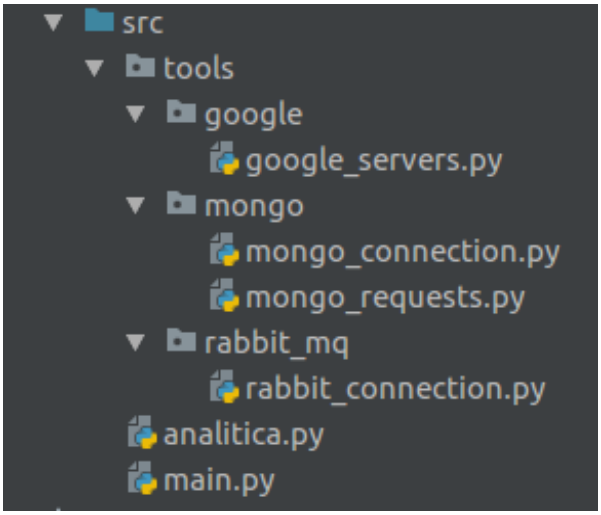

Рисунок 3.6 – Зображення всіх робочих модулів системи

Основний модуль складається з двух файлів analitica.py і main.py, а також під модуля tools, що зберігає в собі засоби роботи:

- Базою даних mongodb;
- Системою обміну повідомленнями RabbitMQ;
- Засобами управління Google Cloud Platform.

Детальний зміст під модулів та файлів буде розглянутий у 3 частині.

3.2.3 Модуль deployment_config

Модуль deployment config, придназначений для зберігання паролів та ключів, а також системних утиліт, що дозволяють розгортати програму у хмарному середовищі Google Cloud Platform. На рисунку 3.7 частково показана структура цього модуля.

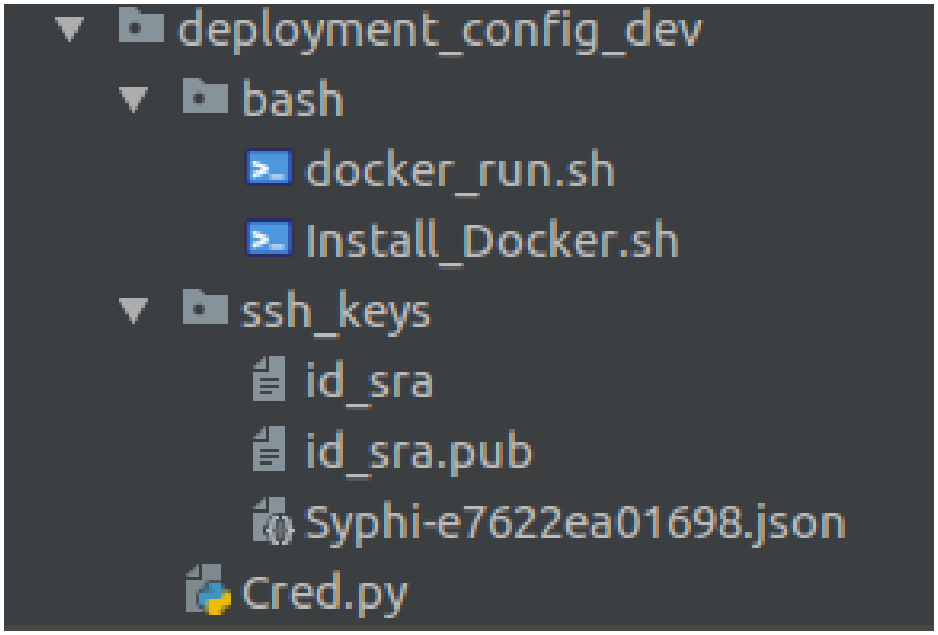

Рисунок 3.7 – Структура модуля deployment config

Цей модуль містить в собі декілька директорій і один файл. У файлі Cred.py знаходяться паролі і ключі для підключення до бази даних, системи обміну повідомленнями, а також назви її черг. Отже всі паролі знаходяться в зручному місці для їх зміни, і можуть бути завантаженні в проект через DNS сервер, що збільшує безпеку даної системи. Також у модулі ssh_keys, знаходяться згенеровані ssh-ключі для доступу до створених, у ході роботи системи, серверів. А також основний ключ від аккаунту Google Cloud Platform, за допомогою якого забезпечується, безпосередня робота з віртуальними серверами, а саме створення, редагування і видалення.

Також тут знаходиться директорія bash, в якій зберігаються всі bash скрипти. Bash - це інтерфейс командного рядка для взаємодії з операційною системою. Він широко доступний для всіх інших систем. За

допомогою таких скриптів дуже легко автоматизувати роботу системи. Тобто, це послідовний набір виконання команд, командного рядку операційної системи. Наявні 2 bash скрипти. Перший встановлює на операційну систему додаток docker а також docker-compose. Інший будує зображення контейнера а потім запускає контейнер.

3.2.4 Модуль project_storage

Це один із чотирьох модулів системи, що відповідає за збереження вихідного коду модулів, що працюють в системі. Тобто тут зберігаються всі необхідні файли для модулів, що працюють на хмарній платформі google cloud platform, а також використовують систему обміну повідомленнями RabbitMQ. Розглянемо цей модуль на прикладі структури проекту progect_B, що представлений у даній магістерській дисертації, а структура якого наведена на рис. 3.8.

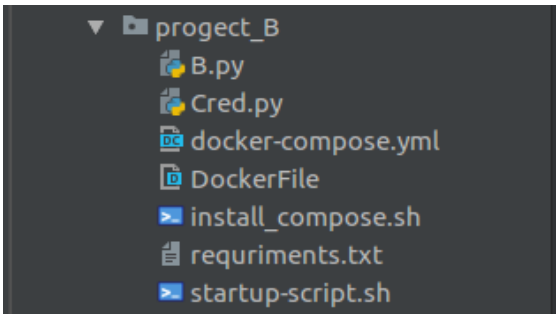

Рисунок 3.8 – структура проекту progect_B

На прикладі структури, що зображена на рис. 3.8, можна виділити наступні компоненти:

- Код безпосередньо виконуючої програми, що знаходиться в файлі B.py;
- Паролі, та інші системні налаштування у файлі Cred.py;
- Install_compose.sh, bash скрипт, який встановлює на систему Docker а також docker-compose;
- DockerFile, з якого створюється образ системи;
- Файл з переліком обов’язкових до встановлення пакетів та модулів - requriments.txt;
- Docker-compose.yml, з якого будуються та запускаються образи системи.

Якщо розглянути всі компоненти системи більш детально, то треба зупинитися на B.py. Потрібно, щоб в цьому файлі містилися деякі структури і функції, що дозволяють прослуховувати чергу, в системі RabbitMQ. Саме в цьому файлі таку функцію виконують наступні структури.

```
import pika
channel.queue_declare(queue=Config.B)
channel.basic_consume(callback, queue=Config.B)
channel.start_consuming()
```

Спочатку треба імпортувати модуль для роботи з системою обміну повідомлень, за допомогою засобів мови програмування python 3. Далі йде рядок, де йде створення черги, якщо такої ще не існує. Для цього потрібно лише ім'я черги. В наступному рядку йде присвоєння стартової функції, для черги з ім’ям Config.B. Отже кожне повідомлення, що буде отримане з черги с таким ім’ям, буде оброблено функцією callback, що представляє з себе наступне.

```
def callback(ch, method, properties, body):
  main(body)
  ch.basic_ack(delivery_tag=method.delivery_tag)
```

Для кожної задачі, що надходить з черги, буде виконана функція main, в яку параметром передають тіло повідомлення, що надійшло. Після виконання основної функції, буде визваний метод, що помічає поточну задачу як виконану.

DockerFile описує структуру системи, потрібну для виконання програми. Отже спочатку встановлюється система ubuntu 16.04, так як встановлюється її повна версія, що містить в собі засоби графічного відображення, що є зайві. Отже можна відключити і видалити їх, що виконується строчкою RUN DEBIAN_FRONTEND=noninteractive. Далі йде повне копіювання всіх файлів у цьому модулі до зображення системи, та встановлення головної директорії у зображенні системи. Це виконується завдяки:

```
COPY . /app
WORKDIR /app
```

Після цього встановлюються локалі, які підтримують локальні технології, використовують змінні середовища для визначення угод, використовуваних для форматування дати і часу, відображення символів, відображення валюти і вибору кодової сторінки.

```
ENV LC_ALL en_US.UTF-8
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US.UTF-8
RUN apt-get update && \
  apt-get -y install apt-utils locales && \
  echo "en_US.UTF-8 UTF-8" >> /etc/locale.gen && \
  locale-gen en_US.utf8 && \
  /usr/sbin/update-locale LANG=en_US.UTF-8
```

Останній етап, це встановлення всіх необхідних залежностей мови python 3, за допомогою pip - системи управління пакетами, яка використовується для установки і управління пакетами програмного забезпечення. Всі залежності знаходяться у файлі requriments.txt, в якому вказана назва пакету, а також його версія. Останнє, що міститься в цьому файлі, є інструкція до виконання при початку роботи системи. Вона виглядає наступним чином CMD ["python3", "B.py"], отже запускається пітон файл B.py.

Для наступного динамічного розширення системи, для запуску контейнеру, використовується утиліта docker-compose, що дозволяє одночасно запускати декілька контейнерів, і чітко регламентує їх налаштування.

Отже модуль project_storage має все необхідне для роботи системи, для розгортання проекту на сервері, за допомогою технології docker. Також при створенні віртуальної машини, google cloud platform надає можливість з початку роботи запустити на сервері контейнер, але було вирішено відмовитися від такого варіанту, адже тоді було б неможливо запускати кілька контейнерів на віртуальній машині одночасно.

3.3 Опис роботи системи

Система починає працювати з того, що задаються деякі глобальні змінні, для оптимізації роботи системи. Треба задати деякі параметри, які будуть унікальні для кожного користувача, а саме:

- Ідентифікаційний ключ, що надається Google Cloud Platform, в якому вказані наступні унікальні поля - "project_id", "private_key_id", "private_key", "client_id", що надаються кожному користувачу.
- Максимальна кількість серверів, для одного слухача черги.
- Залежність від слухача черги і проекту, що слухає саме цю чергу.
- Потужності віртуальних машин.

Система починає виконуватись з попереднього налаштування усіх модулів, та залежностей програми. Після цього починає безпосередньо працювати система розподілення навантаження у хмарній інфраструктурі. Перш за все встановлюється з’єднання з базою даних mongodb, за допомогою модуля pymongo. Встановлення з'єднання реалізовано у модулі src/tools/mongo/mongo_connection.py. В ньому знаходяться 3 функції, mongo_connection() - що безпосередньо виконує встановлення, з'єднання і в випадку якщо це не виходить вертає значення False, або повертає значення курсору бази даних. Наступна реалізована функція mongo_check(), що приймає як аргумент курсор, що був отриманий під час з’єднання, і якщо з'єднання активне повертає значення True, і навпаки False коли з'єднання розірване. Останньою функцією є повторне отримання курсору, якщо попередній зв’язок був розірваний.

Отже встановлюється зв'язок з базою даних системи, сама через подібні рядки коду.

```
mongodb = mongo_connection.mongo_connection()
if not mongo_connection.mongo_check(mongodb):
mongodb = mongo_connection.mongo_reconnect(mongodb)
```

Після встановлення з’єднання з базою, треба імена всіх черг у RabbitMQ, для подальшого відстежування, і занести у базу даних. І після цього починається нескінченний цикл, що буде моніторити стан системи. Його місія полягає у тому, щоб для кожної черги, в будь який момент часу визначати, чи достатня кількість копій віртуальних машин є слухачами черги, для того щоб забезпечити максимальну швидкість виконання, і мінімальну швидкість відповіді. Отже коли система проходить по всіх чергах, вона збирає актуальну інформацію, щодо кількості слухачів а також кількості завдань у черзі. Так як система постійно моніторить стан зада, вона є динамічною. Постійно відомі дані про кількість задач і слухачів в 4 попередні ітерацій. Отже система приймає рішення про завантаженість черги на основі попередньої завантаженості. Маючи 4 попередні стани черги, і отримуючи одну нову, треба визначити, чи справляються слухачі з навантаженням. Для такої системи існую 3 стани для слухачів:

- стан критичного навантаження;
- спокійний стан;
- стан надмірного використання ресурсів.

Характеристика стану критичного навантаження, це коли сервери не взмозі справлятися з кількістю задач в черзі. І для підтримання швидкодії виконання потрібно створювати нові сервери, що будуть долучатися, і починати відпрацьовувати задачі з деякої черги.

Спокійний стан характеризується задоволеністю швидкодією задач, що не потребує додаткових ресурсів, отже до такого стану треба намагатися отримати.

Стан надмірного використання ресурсів, характеризує себе тим, що немає фізичної потреби у великій кількості серверів, а отже їх можна зупинити чи видалити.

Характеристика стану надається за допомогою кількості задач у черзі. Якщо кількість тримається на 0, за попередні 4 ітерації, то можна

стверджувати, що система зашла у стан надмірного використання ресурсів. А отже їй не потрібно така кількість віртуальних машин, що отримують задачі з даної черги, і їх можливо видалити. Якщо кількість зменшення задач менше ніж 25% від кількості задач на попередній ітерації, то можна сказати, що система знаходиться у спокійному стані. З цього слідує, що кількість віртуальних машин змінювати не потрібно. А отже якщо ситуація не підходить під попередні, то можна стверджувати, що в системі стан критичного навантаження, що потребує збільшення кількості віртуальних машин, що будуть обслуговувати чергу. Для цього буде визначений проект, що прослуховує дану чергу, далі буде розгорнутий додатковий сервер, в межах одного проекту, що обслуговується google cloud platform. Після цього програмно встановлюється ssh з'єднання, і через нього передаються всі існуючі файли проекту. Далі виконується команда системної консолі, що запускає проект, встановлює його на виконання задачі і на прослуховування деякої черги. Таким чином система створює додаткову машину, і забезпечує більш швидке виконання задач, за рахунок системних ресурсів.

Отже система працює для мікросервісної архітектури, де модулі використовують для передачі інформації систему обміну повідомленнями RabbitMQ. Також важливо щоб усі модулі використовували систему для управління ізольованими Linux-контейнерами Docker, адже він спрощує процес розгортання модуля на віртуальній машині. Спочатку будуються та завантажуються всі залежності, які прописані в Dockerfile, що містить в собі всі потреби системи, що запускається в контейнері, а після цього виконується його запуск. В даному алгоритмі не так важливо що саме буде виконувати модуль, адже для такої системи найбільш важливим є те, що він забирає задачі с черги.

Алгоритм розгортання зображений на рис. 3.9, де показані ключові етапи роботи алгоритму.

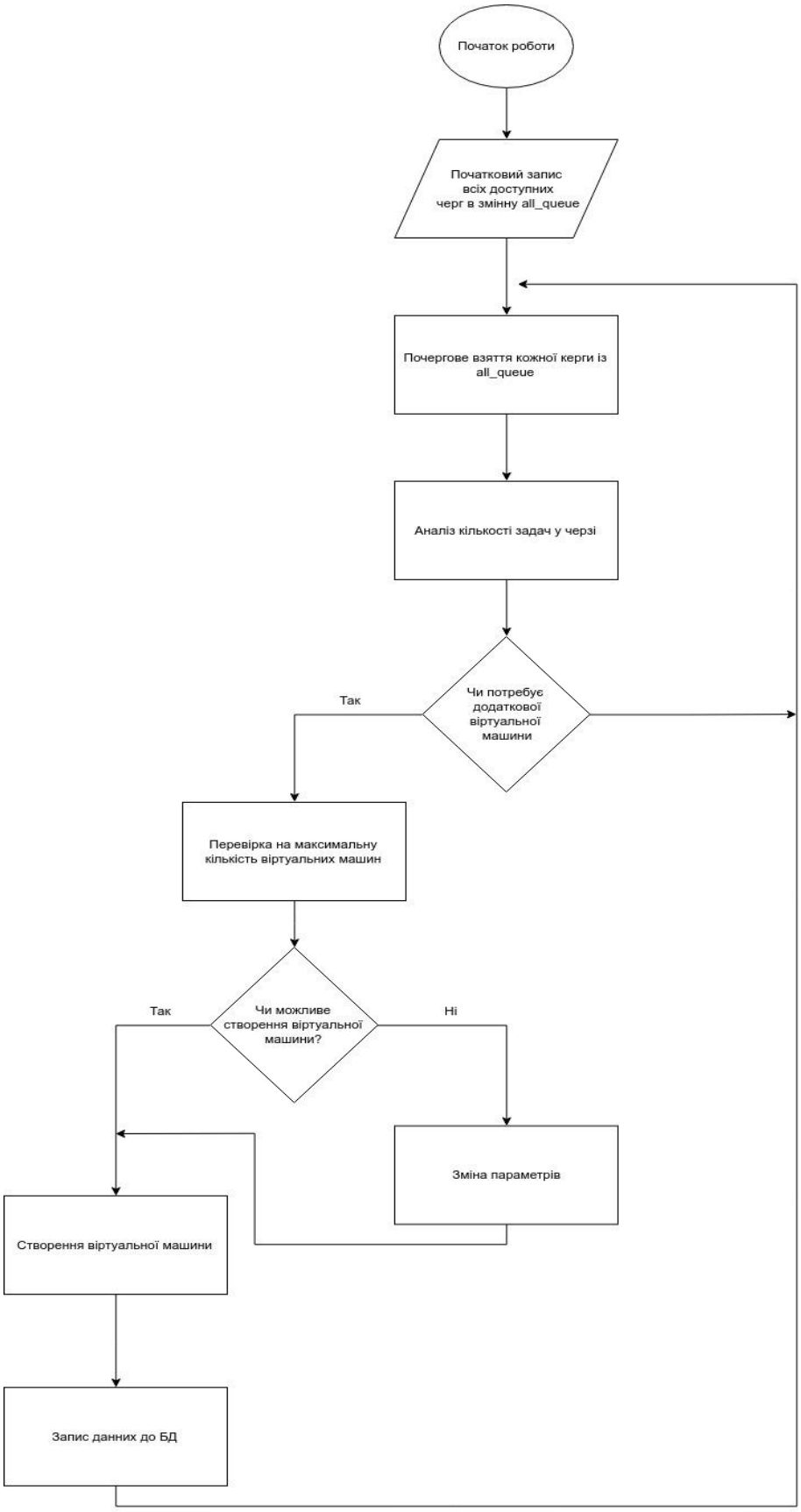

Рисунок 3.9 - Основні етапи роботи системи розподілу навантаження в хмарній структурі

На рис. 3.9 зображений узагальнений план роботи системи, але для повного розуміння роботи системи, треба детальніше розглянути деякі її аспекти. Наприклад аналіз завантаженості черги в RabbitMQ, або розгортання нового серверу алгоритм якого показаний на рис. 3.10.

Рисунок 3.10 – Алгоритм створення і налаштування віртуального серверу

Створення нового серверу починається з генерування ssh ключа до цього серверу. Secure Shell, або SSH - це криптографічний мережевий протокол, який дозволяє користувачам безпечно виконувати ряд мережевих сервісів по незахищеній мережі. SSH забезпечує більш

безпечний спосіб входу на сервер з SSH, ніж використання тільки пароля. У той час як пароль може бути зламаний за допомогою грубої сили, ключі SSH майже неможливо розшифрувати тільки грубою силою [14].

Створення пари ключів дає вам дві довгі рядки символів: відкритий і закритий ключ. Ви можете розмістити відкритий ключ на будь-якому сервері, а потім розблокувати його, підключившись до нього з серверу-клієнта, у якого вже є закритий ключ. Коли вони збігаються, система розблокується без необхідності пароля. Ви можете підвищити безпеку ще більше, захищаючи закритий ключ фразою-паролем [10].

Створення ssh ключа, на клієнтській машині виконується в декілька кроків. Перший, це команда, яка створює ключову пару RSA.

```
ssh-keygen -t rsa
```

Наступний крок, це збереження ключа у потрібній директорії. Це відбувається наступним чином, коли введено попередню команду, потрібно відповісти ще на декілька питань, а саме куди зберігати ключ, а також задати пароль для ключа. Це залежить від вас, чи хочете ви використовувати кодову фразу. Введення кодової фрази має свої переваги: безпека ключа, незалежно від того, як він зашифрований, все ще залежить від того, що його не видно нікому іншому. Якщо закритий ключ, захищений паролем, потрапляє в неавторизоване володіння користувачами, вони не можуть увійти в пов'язані з ним акаунти, поки не з'ясують кодову фразу. Єдиний недолік, звичайно ж, в тому, що у вас є парольний фраза, потрібно набирати її кожен раз, коли ви використовуєте пару ключів.

Так як система виконує створення серверів, а також ключів у автоматичному режимі, тому потрібно змінити команду створення, щоб уникнути вводу з клавіатури директорії збереження ключа і фрази пароля. Отже потрібно використовувати наступну команду.

```
ssh-keygen -t rsa -N "" -f my.key
```

-N "" говорить, що він використовує порожню парольну фразу (таку саму, як дві з вхідних в інтерактивний сценарій). -f my.key повідомляє, що він зберігає ключ у my.key. Або можливий інший варіант, коли можливо поставити знак переносу строки у інтерактивну консоль.

```
echo -e "\n\n\n" | ssh-keygen -t rsa
```

Наступним етапом є створення ssh з'єднання с сервером. Виконання такого з'єднання виконується завдяки таких рядків коду.

```
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

par_dir = os.path.abspath(os.path.join(__file__, os.pardir, os.pardir))
_keys_file = os.path.join(par_dir, "deployment_config", "ssh_keys", "id_sra")

_keys_file = open(_keys_file, 'r').read()
_keys_file = io.StringIO(_keys_file)
_vm_key = paramiko.RSAKey.from_private_key(_keys_file, password="revo1917")
ssh.connect(hostname=_ip, port=22, username="ralko96", pkey=_vm_key)
```

Secure Shell (SSH) забезпечує відкритий протокол для забезпечення мережевої комунікації який є менш складним і дорогим, за апаратні рішення VPN. Клієнт/сервер архітектура Secure Shell забезпечує оболонку, передачу файлів і тунелювання даних через TCP/IP додатки. З'єднання SSH забезпечує надійну аутентифікацію, шифрування та цілісність даних для боротьби з крадіжкою паролів та інших загроз безпеці. Клієнти і сервери – це реалізації версії Windows, що сумісні з програмним забезпеченням SSH на інших платформах.

Функціональність Secure Shell надає три основні можливості, які відкривають двері для багатьох дій безпечні рішення:

- захист командної оболонки;
- захист передачі файлів;
- переадресація портів.

Командні оболонки, такі як доступні в Linux, Unix, Windows операційних системах надають можливість швидкому виконанню програм

та іншим командам. Захищена командна оболонка або віддалений доступ дозволяє редагувати файли, переглядати вміст каталогів та мати доступ до спеціальних додатків баз даних. Адміністратори систем та мереж можуть віддалено запускати, починати, переглядати або припиняти послуги та процеси, створювати облікові записи користувачів та змінювати доступ до файлів та каталогів.

Переадресація портів – це потужний інструмент, який забезпечує безпеку для програм які працюють з TCP/IP-протоколом включаючи електронну пошту, бази даних, а також внутрішні програми. Переадресація портів іноді називається тунелюванням, що дозволяє отримувати дані, як правило, необов'язкові. Після налаштування перенаправлення на порт, Secure Shell переміщує трафік з програми (як правило, клієнт) і надсилає його через зашифрований тунель, потім передає його в програму з іншого боку (зазвичай на сервері). Кілька програм можуть передавати дані по одному мультиплексному каналу, усуваючи необхідність відкривати додаткові порти в брандмауері або маршрутизаторі.

Для деяких програм захищена командна оболонка не є безпечним захистом. Можна використати можливості перенаправлення портів Secure Shell. Потрібно створити зашифрований тунель, в якому може бути запущена програма, як показано на рис. 3.11. Віртуальний мережевий клієнт - це приклад програми дистанційного керування для графічного інтерфейсу на платформі.

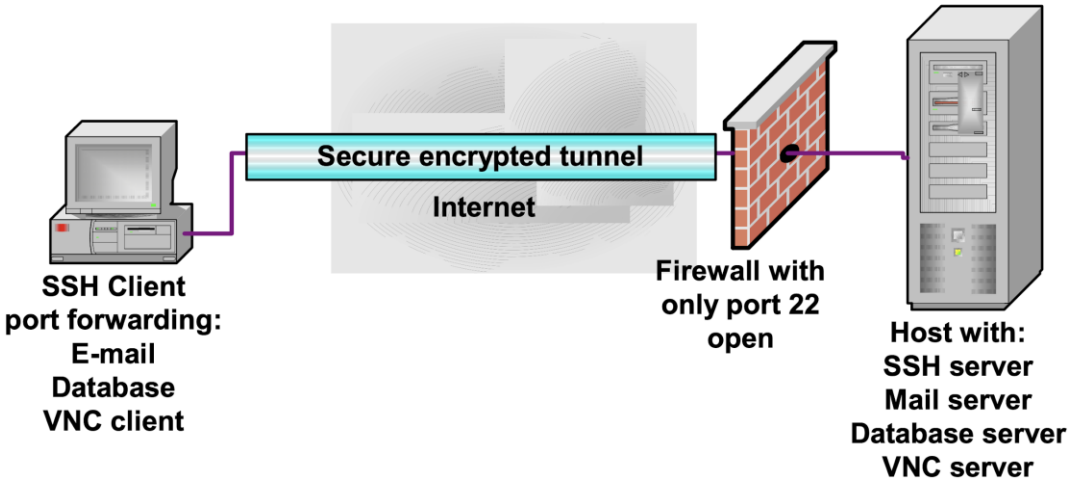

Рис. 3.11 – Перенаправлення портів дозволяє декільком додаткам TCP / IP, щоб спільно використовувати одне захищене з'єднання.

Протокол Secure Shell надає чотири основні переваги безпеки:

- аутентифікація користувача;
- автентифікація хоста;
- шифрування даних;
- цілісність даних.

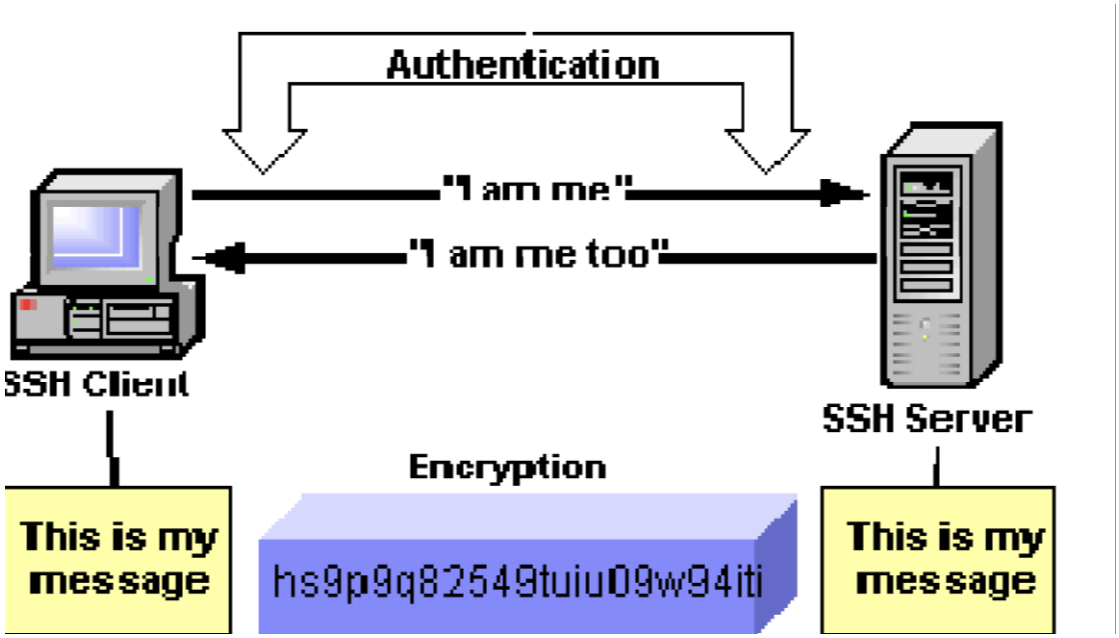

Рисунок 3.12 - Кодування повідомлення між клієнтом і сервером.

Автентифікація користувача також ще називається ідентифікатором користувача та є засобом перевірки системи. Доступ надається лише призначеним користувачам, а всім іншим - відмова. В наш час багато хто використовує методи аутентифікації, починаючи від паролів захисту до більш надійні механізмів безпеки, як показано на рис. 3.13. Більшість реалізацій Secure Shell включають в себе методи авторизації паролів та відкритих ключів.

Якщо ім'я та пароль вказані при аутентифікації збігаються з ім'ям користувача та паролем, збереженим на віддаленій системі, аутентифікації завершується успішно та на система надає доступ доступ. Деякі протоколи, як FTP та Telnet, відправляють імена користувачів і паролі як легко помітний ASCII текст, що дозволяє за допомогою програми зафіксувати будь-кого, а потім отримати доступ до системи.

Хоча паролі не вимагають додаткової конфігурації або налаштування для користувачів, вони за своєю суттю вразливі, тому що про них може дізнатися кожен, і пароль може потрапити у будь-яку вразливу систему. Через це рекомендується комбінувати або замінити аутентифікацію пароля за допомогою іншого методу, такого як відкритий ключ.

Аутентифікація за допомогою відкритого ключа є одним із найбільш безпечних методів аутентифікації за допомогою Secure Shell. Аутентифікація з відкритим ключем використовує пару комп'ютерних ключів - один загальнодоступний та один приватний. Кожна клавіша зазвичай має довжину від 1024 до 2048 біт.

Загальнодоступні ключі зазвичай створюються за допомогою утиліти генерації ключів. Обидва ключі утворюються одночасно в парі і, хоча два з них пов'язані, приватний ключ не може бути обчислений за допомогою відповідного відкритого ключа. Щоб отримати доступ до облікового запису на сервері Secure Shell, обліковий ключ клієнта повинен бути завантажений на сервер. Коли клієнт підключається до сервера, він

повідомляє, що має секретний або приватний аналог відкритого ключа на цьому сервері, і доступ надається.

Зазвичай, приватний ключ має "парольну фразу", навіть якщо секретний ключ викрадено, атакуючий повинен вгадати парольну фразу, а потім вже отримати доступ. Аутентифікація з відкритим ключем не довіряє ніякій інформації від клієнта та не надає доступ, поки клієнт не зможе довести, що він має "секретний" приватний ключ.

Secure Shell Agent – це спосіб автентифікації на декількох серверах безпеки, який визначає відкритий ключ клієнта без потреби повторно вводити парольну фразу кожного разу. Крім того, включивши переадресацію агента, можна підключитися до мережі Secure Shell-серверу, усуваючи необхідність компрометації цілісності вашого приватного ключа.

Таке з’єднання встановлюється для того, щоб виконувати команди для будування зображення системи, а також для виконання її запуску. Також початкове ssh з’єднання встановлюється для виконання sftp з’єднання.

SSH2 представив більш надійний метод безпечної передачі файлів: Secure Shell File Transfer (SFTP). SFTP використовує захищену оболонку для аутентифікований, зашифрованої передачі файлів. SFTP забезпечує функціональність звичайного FTP без ризиків, пов'язаних із запуском незахищених FTP-протоколів. Заміна FTP за допомогою SFTP може значно зменшити вразливість файлового сервера. Крім того, SFTP не ускладнюється архітектурою FTP-мультизастосування. Як показано на малюнку 2, SFTP захищає всі бітові імена користувачів, паролі, списки та файли, передані між клієнтом SFTP і сервером.

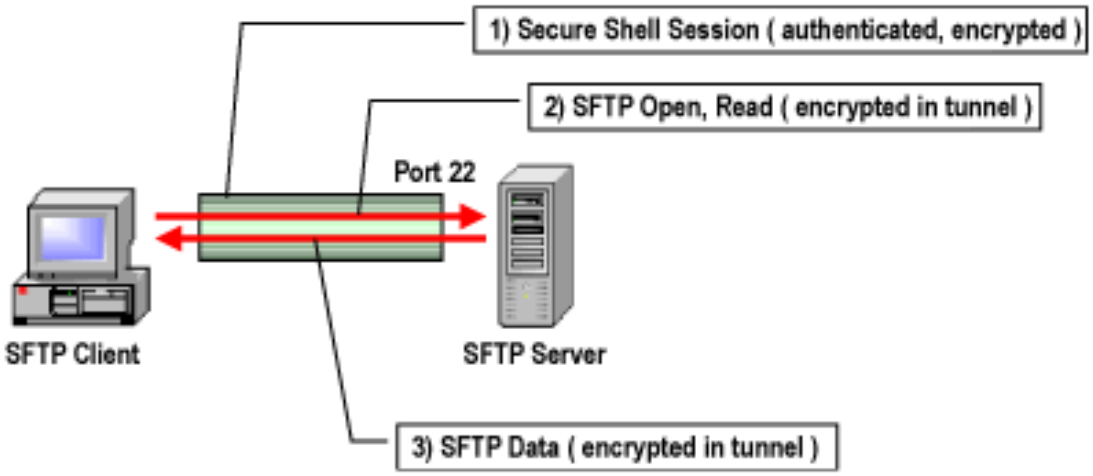

Рисунок 3.13 - Відкриті SFTP, команди читання (тунель у сеансі SSH2)

SFTP не використовує перенаправлення портів. Замість цього SFTP працює як підсистема, інтегрована з SSH2. Клієнт SFTP ініціює сеанс Secure Shell для цільового SFTP-сервера. Протокол SFTP складається з команд віддаленої файлової системи, ці команди передаються безпосередньо через існуючу сесію Secure Shell. Підмножина SFTP також забезпечує основу для SCP (2), заміни SCP, що передається порту.

Для кінцевого користувача SFTP та SCP (2) виглядають дуже схожими на застарілі способи передачі файлів, які вони замінюють. Для максимальної сумісності підтримує як захищені, так і застарілі способи передачі файлів. Якщо це можливо, організації повинні використовувати SFTP – найбільш надійний спосіб безпечного пересилання файлів через Secure Shell.

Secure Shell може бути використана для безпечного адміністрування системи, як показано на малюнку 3.14. Замість чистого тексту Telnet, багато адміністраторів віддають перевагу використанню клієнта Secure Shell для віддаленого входу до системи або доступу до командної оболонки. Secure Shell, такі як OpenSSH, зазвичай присутні на серверах UNIX і все частіше зустрічаються на мережевих пристроях, таких як маршрутизатори, комутатори та брандмауери.

Багато завдань з адміністрування є інтерактивними, але повне рішення завдань вимагає безпечної передачі файлів. Системні адміністратори повинні передавати програмне забезпечення, файли конфігурації, дані облікового запису користувача та записи про використання. FTP через Secure Shell захищає пароль адміністратора - необхідний для віддаленого адміністрування через загальнодоступний Інтернет. SFTP йде на крок далі, захищаючи цінний і чутливий файловий вміст. Наприклад, перенесення облікових записів за допомогою SFTP запобігає несанкціонованому розголошенню номерів кредитних карток, дозволів та паролів. Крім того, це доводить, що виконання кроків для забезпечення конфіденційності, потенційно обмежує відповідальність.

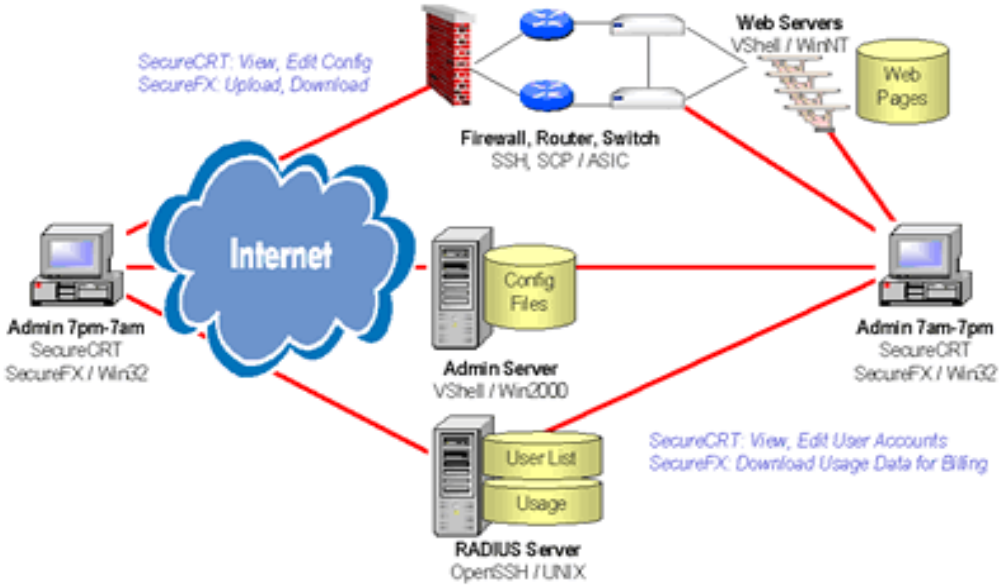

Рисунок 3.14 - Безпечне адміністрування системи

SFTP може бути використаний для надійного передавання файлів всередині та між підприємствами. Розгортання серверів SFTP у стратегічних локальних мережах створює міжплатформенну інфраструктуру обміну файлами для взаємодії та передачі робочих продуктів бізнес-підрозділам, клієнтам та партнерам. У цьому прикладі підприємство використовує SFTP для передачі фінансових електронних

таблиць зовнішньому аудитору та замовлення на придбання виробника. Інтернет-постачання підвищує ефективність бізнесу, однак авторизованим користувачам слід надати доступ до цих файлів. Поєднуючи паролі та аутентифікацію з відкритим ключем, ця компанія перевіряє ідентифікацію одержувача, перш ніж надсилати будь-який файл. Використовуючи MAC для виявлення модифікації, одержувачі запевняють, що скопійовані файли залишаються аутентифікованими.

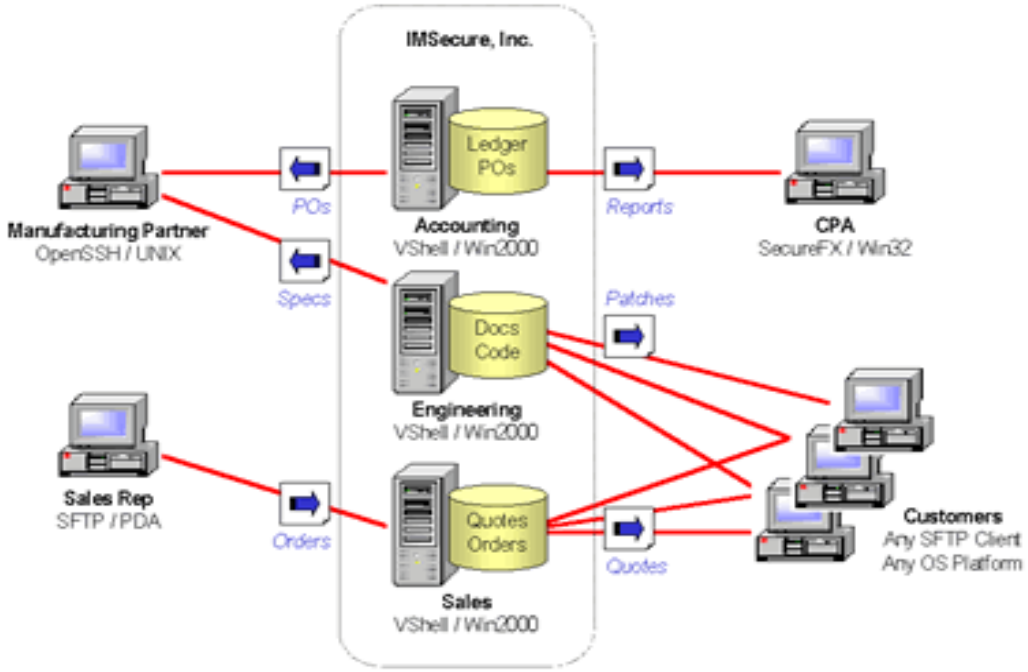

Рисунок 3.13 - Захист бізнес-бізнесу

У транзакціях між бізнесом організації, як правило, не можуть розголошувати операційну систему, сервер або програмне забезпечення клієнта. Наприклад, якщо розглянути консультантів, які надають конфіденційні звіти та компанії з інформаційних технологій, які надають клієнтам не повні версії програмного забезпечення. Ці ситуації вимагають незалежного від платформи рішення, яке може бути розгорнуто швидко, при мінімальних інвестиціях, розміщення будь-якого клієнта, як на рис. 3.13. Передача файлів на основі Secure Shell цілком підходить, тому що

програмне забезпечення є доступним майже для кожної операційної системи, і проблеми сумісності є відносно рідкісним явищем.

У бізнеса існує мотивація для захисту конфіденційності переданих файлів, законодавство є фактором, що набуває все більшого значення. У США федеральне, штатне та місцеве самоврядування прийняли закон про конфіденційність, вимагаючи від бізнесу визначити політику, яка обмежує розкриття особистої інформації. Директива Європейського Союзу з захисту даних вимагає, щоб передача інформації третій країні здійснювалась тільки за певних умов. Ця директива застосовується не лише до ЄС, але і до будь-якої компанії, що веде бізнес з європейськими громадянами.

Деякі закони про конфіденційність виділяють окрему галузь. У США один із прикладів - Gramm Leach Billey (GLB). З метою посилення конкуренції в галузі фінансових послуг, GLB включає положення, яке вимагає захисту приватних осіб. Федеральна резервна система, національні банки та ощадні асоціації не є єдиними організаціями, що зазнали впливу. Іпотечні компанії та страхові андеррайтери також включені. Під GLB фінансові установи повинні встановлювати відповідні заходи безпеки та конфіденційності для записів клієнтів - зокрема, запобігання несанкціонованому розголошенню особистої інформації. Відповідність GLB починається з визначення політики; SFTP - це один із інструментів для реалізації цих стратегій.

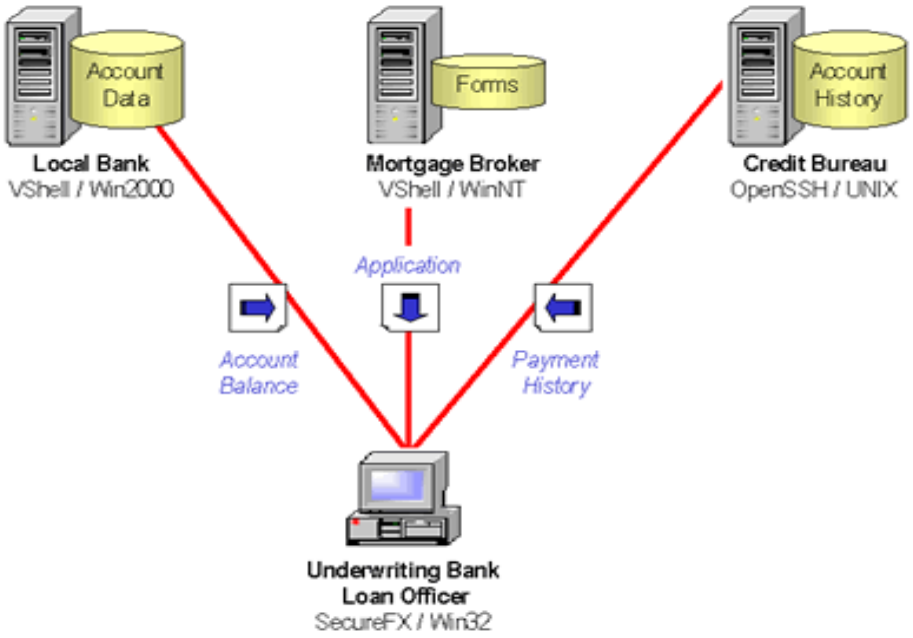

Рисунок 3.14 – Захист файлів, що розповсюджуються між фінансовими установами

Наприклад, SFTP може забезпечити сильну автентифікацію та доступ до приватних даних, що беруть участь у затвердженні іпотечного кредиту, за участю декількох компаній. Як показано на малюнку 3.14, андеррайтинговий банк використовує SFTP, щоб залучати заявки на отримання кредитів з бази даних іпотечного брокера, отримати історію з кредитного бюро та перевірити залишки на рахунках. У цьому прикладі SFTP забезпечує цілісність та конфіденційність особистої інформації під час транзиту між співпрацюючими фінансовими установами. Журнали подій сервера забезпечують аудит, ідентифікуючи, хто і коли отримує доступ. Звичайно, SFTP необхідно поєднувати з додатковими заходами безпеки, захищаючи дані, що зберігаються в кожній фінансовій установі.

Охорона здоров'я – це інша галузь, яка суттєво вплинула на нове законодавство про приватність. Закон про переносимість та відповідальність за охорону здоров'я (HIPAA) 1996 року був створений для полегшення потоку медичної інформації при захисті конфіденційних даних пацієнтів від невідповідного доступу, розкриття та використання. Правила HIPAA визначають код та форми транзакцій, права на конфіденційність,

інформаційну безпеку та ідентифікатори для пацієнтів, постачальників, планів та роботодавців. Вимоги до безпеки HIPAA охоплюють адміністративну політику та процедури, фізичні гарантії, технічні послуги та технічні механізми. Технічні служби охоплюють "дані в спокої". Технічні механізми охоплюють "дані в русі", що вимагають автентифікації суб'єкта, контролю доступу, шифрування, цілісності даних, звітування про події та сигналізації. SFTP, безумовно, є корисним інструментом для реалізації політики, яка відповідає вимогам безпеки HIPAA.

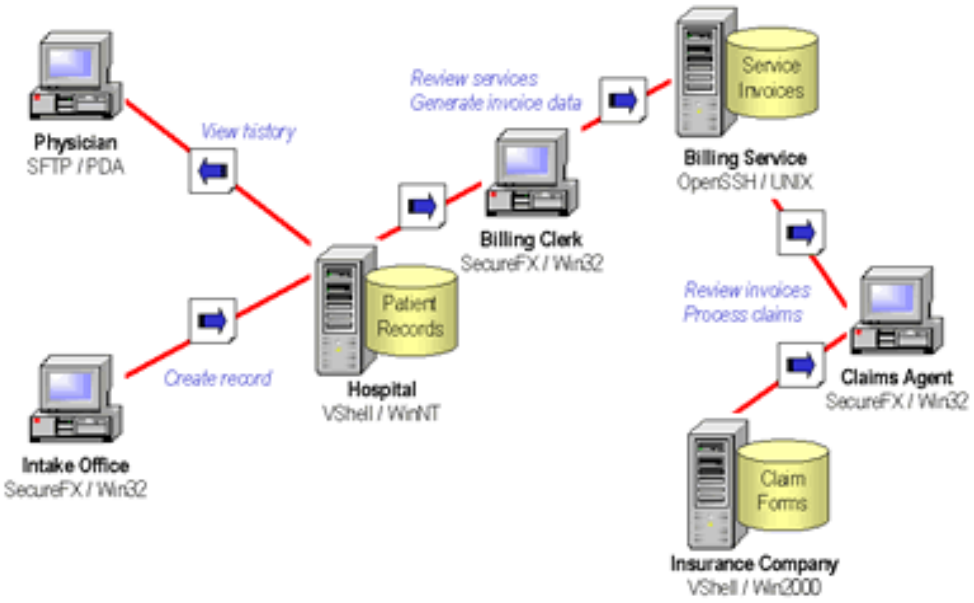

Рисунок 3.15 - Захист реєстру пацієнтів в охороні здоров'я

На малюнку 3.15 показано, як SFTP може використовуватися, як технічний механізм, захищає дані у розподіленій системі охорони здоров'я. Організації включають медичних працівників (медиків, лікарень), плани охорони здоров'я (страхові компанії, ЗМО, Medicare), клірингові будинки (білінгові послуги, компанії з обробки претензій) та будь-який інший бізнес-партнер, залучений до "мережі довіри". У цьому прикладі Secure Shell MAC перешкоджає зміні повідомлення. Шифрування Secure Shell забезпечує конфіденційність інформації про охорону здоров'я. Журнали подій сервера надають інформацію, необхідну для полегшення аудиту

безпеки. HIPAA також вимагає, щоб лікарі мали швидкий доступ до запитів пацієнтів, створених іншими користувачами. Захищений електронний доступ за допомогою стандартного протоколу, як SFTP, відповідає цій вимозі.

Отже за допомогою цього протоколу передачі даних, всі необхідні файли системи передаються на сервер.

```
sftp = ssh.open_sftp()
sftp = sftp.put(os.path.join(sftp_dir, element), "/home/ralko96/app/" + element)
```

Таким чином всі потрібні для проекту файли переносяться на віддалений сервер. Можливий ще один варіант переносу файлів на віртуальну машину, за допомогою спільного диску, але при такому варіанті завантаження проекту повільне, тому від нього було вирішено відмовитись.

Після переносу всіх файлів системи, виконується команда docker-compose, яка виконує відразу дві функції.

```
sudo docker-compose -f /home/ralko96/app/docker-compose.yml up --build
```

По-перше за цією командою починається побудова зображення системи, з вказаного файлу docker-compose.yml. А відразу після цього починається побудова системи, з усіма залежностями, які описані у yml файлі.

Після повного запуску системи, відбувається закриття ssh з’єднання, і після цього система працює на віртуальній машині.

3.4 Результати роботи системи

Результати роботи системи розподілу навантаження у хмарній інфраструктурі показані в таблиці 3.2, тоді як у таблиці 3.1 показані результати роботи, коли не використовується ніяка інша система.

Таблиця 3.1 – Таблиця залежності кількості виконання задач від часу, в системі без розподілу навантаження

		Кількість виконаних задач			
Назва черги		A	B	C	Загальна
Час роботи системи (хвилини)	5 хв.	8	12	10	30
	10 хв.	17	24	21	62
	15 хв.	26	36	32	94

Таблиця 3.2 – Таблиця залежності кількості виконання задач від часу, в системі де присутній алгоритм розподілу навантаження і його реалізація

		Кількість виконаних задач			
Назва черги		A	B	C	Загальна
Час роботи системи (хвилини)	5 хв.	15	20	16	51
	10 хв.	30	40	33	103
	15 хв.	45	60	50	155

Отже з таблиць видно кількісну різницю виконаних завдань. На часовому проміжку у 5 хвилин, кількість додатково оброблених завдань складає 21 задача, на проміжку у 10 хвилин - 41 задача, а на 15 хвилинах

61 задача. На таблиці Таблиця 3.2 візуально представлено кількість задач, що додатково оброблені а отже і видно користь алгоритму і системи розподілу навантаження у хмарній інфраструктурі.

Висновки до розділу 3

Розроблено систему розподілу навантаження на сервіси хмарної інфраструктури на основі модифікованого способу динамічного балансування навантаження з використанням таких технологій як: Python3, Google Cloud SDK, RabbitMQ, Docker та інших.

Для реалізації системи розподілу навантаження використовується модифікований спосіб динамічного балансування навантаження з масштабуванням на хмарній інфраструктурі, який дозволяє аналізувати та враховувати кількість задач в кожній черзі RabbitMQ, що збільшує продуктивність системи.

Проведені дослідження застосування модифікованого способу балансування навантаження з урахуванням ключових етапів роботи та параметрів, заданих користувачем, показали, що такий підхід дозволяє підвищити продуктивність та безпеку системи в цілому.

ВИСНОВКИ

У даній роботі запропоновано систему динамічного балансування навантаження в хмарному інфраструктурі за умови використання модулями системи обміну повідомленнями RabbitMQ, а саме алгоритм динамічного навантаження з урахуванням кількості задач на кожному сервері. Метою даного методу є ефективне використання ресурсів у хмарному середовищі. Експериментальні результати показали зменшення часу обробки.

Було проведено загальний огляд технології хмарних обчислень. Дано визначення хмарним обчисленням та з'ясовані передумови появи хмарних обчислень. Визначені основні види хмарних систем, їх моделі обслуговування та моделі розгортання. Показана важливість систем розподілення навантаження в хмарних обчисленнях. Описані основні способи розподілу навантаження, що використовуються в різних типах хмарних систем. Проаналізовані їх недоліки та переваги, що є передумовою для створення нових рішень в області балансування навантаження.

Запропоновано новий спосіб розподілу навантаження на хмарні системи – обробка статистичних даних про кількість задач, та час їх обробки в реальному часі. Статистичні способи є набагато надійнішими ніж динамічні, які здійснюють розподіл на піку навантаження, так як взагалі не допускають виникнення ситуацій піку навантаження. Статистичні способи зазвичай використовуються в системах з постійною кількістю користувачів, наприклад, в корпораціях, що використовують власні дата-центри. Модифікований спосіб на основі статистичного підходу дозволяє передбачати піки навантаження в реальному часі, аналізуючи дані декількох останніх результатів моніторингу системи та швидко їх ліквідовувати. Така модифікація вирішує проблему

статистичних алгоритмів коли статистичні дані можуть бути викривленими в наслідок постійної зміни кількості користувачів хмарної системи.

Проведено деталізований порівняльний аналіз розповсюджених алгоритмів та методів динамічного балансування навантаження. Зазначено, що балансування навантаження – це оптимізація виконання розподілених або паралельних обчислень за допомогою розподіленої обчислювальної системи. Балансування навантаження передбачає рівномірне навантаження обчислювальних вузлів. При появі нових завдань в черзі RabbitMQ, приймається рішення про те, чи потрібно створювати новий сервер, для прискорення обробки задач.

Представлені результати тестування системи за допомогою власного динамічного способу балансування навантаження. Показано збільшення кількості оброблених задач, і їх залежність від використаних ресурсів.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Дятлов Є.І. Балансування навантаження в розподілених обчислювальних системах / Є. І. Дятлов. - Харків: видання Харків. Ун-та, 2015. – 134 с.

2. Load Balancing in Cloud Computing [Електронний ресурс]. – 2016. Режим доступу до ресурсу: https://www.researchgate.net/publication/297667.

3. The Three Types of Load Balancing You Meet in the Cloud [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://devcentral.f5.com/articles/the-three-types-of-load-balancing-you-meet-in-the-cloud-30704.

4. Bruce Hendrickson, Karen Devine. Dynamic load balancing in computational mechanics// Springer Verlag. – 2003. – P.488

5. Shor R. Cloud Computing for Learning and Performance Professionals / R. Shor, J. Kozloff., 2010.

6. Goscinski A. Cloud Computing: Principles and Paradigms / A. Goscinski, R. Buyya., 2010.

7. Miell I. Docker in Practice, Second Edition / I. Miell, A. Hobson-Sayers., 2018. – 425 с.

8. Load balancing your load balancers for endless scalability [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: http://www.loadbalancer.org/blog/load-balancing-your-load-balancers-for-endless-scalability/.

9. Load Balancing in Cloud Computing: A State of the Art Survey [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://www.researchgate.net/publication/297661797_Load_Balancing_in_Cloud_Computing_A_State_of_the_Art_Survey.

10. Barrett D. SSH, The Secure Shell: The Definitive Guide / Daniel Barrett., 2005.

11. Zaigham M. Cloud Computing: Concepts, Technology & Architecture (The Prentice Hall Service Technology Series from Thomas Erl) / M. Zaigham, R. Puttini, E. Thomas., 2013. – 528 c.

12. Cloud Systems in Supply Chains, 2015. – 319 c. – (Palgrave Macmillan, a division of Macmillan Publishers Limited).

13. Server Load Balancing: Help for Network Administrators, 2001. – 192 c.

14. The Technical Complexities and Risks of Public Key Authentication: The Lack of SSH User Key Management in Large Enterprises Today [Електронний ресурс] // SSH COMMUNICATIONS SECURITY CORP. – 2012. – Режим доступу до ресурсу: https://www.bitpipe.com/detail/RES/ 1337346457_183.html.

15. Fundamental Principles of Network Security [Електронний ресурс] // SSH COMMUNICATIONS SECURITY CORP. – 2012. – Режим доступу до ресурсу: https://www.bitpipe.com/detail/RES/1149185274_283.html.

16. How to ensure app performance, security, and integrity [Електронний ресурс] //RADWARE.– 2018 .– Режим доступу до ресурсу: https://www.bitpipe.com/detail/RES/1541187676_2.html.

17. 5 trends disrupting the hardware load balancing market [Електронний ресурс] //NGINX. – 2018. – Режим доступу до ресурсу: https://www.bitpipe.com/detail/RES/1540303463_222.html.

18. L. Abbott M. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise (2nd Edition) 2nd Edition / M. L. Abbott, M. T. Fisher., 2015. – 624 c.

19. Effective Python: 59 Specific Ways to Write Better Python, 2015. – 256 c.

20. Slatkin B. Learning Python: Learn to code like a professional with Python - an open source, versatile, and powerful programming language / Brett Slatkin., 2015. – 256 c.

21. K. Jones B. Python Cookbook, Third edition 3rd Edition / B. K. Jones, D. Beazley., 2013. – 706 с. – (O'Reilly Media).

22. Harrison M. Treading on Python: Volume 2 Intermediate Python / Matt Harrison., 2013. – 162 с.

23. What's New in MongoDB 4.0 [Електронний ресурс] // MongoDB. – 2017. – Режим доступу до ресурсу: https://www.mongodb.com/collateral/ mongodb-40-whats-new.

24. Become a MongoDB DBA: Bringing MongoDB to Production [Електронний ресурс] // Severalnines. – 2017. – Режим доступу до ресурсу: https://severalnines.com/blog/mongoDB-management-whitepaper.

25. Videla A. RabbitMQ in Action: Distributed Messaging for Everyone 1st Edition / A. Videla, J. J. W. Williams., 2014. – 312 с.

26. P. Kane S. Docker: Up & Running: Shipping Reliable Containers in Production 2nd Edition / S. P. Kane, K. Matthias., 2018. – 352 с.